High Order Partitioned Fully Implicit Runge-Kutta Solvers for Fluid-Structure Interaction

by

Noble T Macfarlane

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Summer 2021

High Order Partitioned Fully Implicit Runge-Kutta Solvers for Fluid-Structure Interaction

Abstract

High Order Partitioned Fully Implicit Runge-Kutta Solvers for Fluid-Structure Interaction

by

Noble T Macfarlane

Doctor of Philosophy in Mathematics

University of California, Berkeley

Professor Per-Olof Persson, Chair

In this work, we develop and analyze a partitioned implicit time integration method designed for fluid-structure interaction (FSI) modelling using discontinuous Galerkin (DG) discretizations and fully implicit Runge-Kutta (IRK) methods. The discontinuous Galerkin method is a high-order finite element method used for fluid simulations on unstructured meshes. We take a partitioned approach, modelling the structures separately and limiting communication between structure and fluid by use of a prediction-correction framework. We verify the high-order accuracy, the stability, and the performance of our method on simple model problems including a one-dimensional sprung piston and a two-dimensional pitching airfoil with prescribed vertical motion and a torsional restoring force, both with structures of two degrees of freedom. Finally we apply our method to the standard problem of a cantilever beam shedding vortexes with two sets of parameters, and a vibrating tuning fork problem. Both these final applications consist of fluids modelled on fully unstructured meshes of high-order triangular elements deformed by radial basis functions according to the structure motion, and structures modeled using a neo-Hookean formulation and discretized using standard continuous finite elements.

Our scheme fully decouples the implicit solutions of the structure and the fluid, with communication limited to boundary conditions and mesh deformation. We present our method with two variations, one using explicit methods for predicted quantities and another using implicit. We implement our method up to seventh order, and compare its cost with standard Diagonally Implicit Runge-Kutta methods where possible. Our findings are that this new method can be significantly cheaper, in particular for the more complex cantilever problem, and it also has better stability properties.

Dedicated to Hannah Lennett

Contents

List of Figures

List of Tables

Acknowledgments

I would like to acknowledge the help and guidance I have received from my advisor, Per-Olof Persson. Your advice has been invaluable in this process. I also want to thank Rocky Sison, who has been an incredible friend and colleague throughout this whole process. Thanks also to Chris Miller. I could not have asked for two better people to share an office with. I also want to thank some other friends in the math department, Anna Siegel, Eric Hallman, Ravi Fernando, Alex Carney, Liz Ferme, and Emily Bain. Having such a community of math students was invaluable.

I would like to acknowledge all the support I got at Mathematics Group at Lawrence Berkeley National Laboratory. Robert Saye and Matt Zahr were so helpful when I wandered into their offices with questions. I also want to thank some of Per's other students. Thank you to Andrew Shi, for the friendship and camaraderie teaching. I also want to thank Will Pazner, whose help was integral in this endeavour. The research he did has served as a foundation for my own, but he was also there to help me find my way in the world of numerical analysis.

I want to thank the people who got me into this and helped me through. Adam Boocher showed me what I was getting myself into. Brad Froehle has always been there when I need him most, from having coffee with me as an informational interview before I joined the math department to the excellently documented code he wrote for all who follow in his footsteps as Per's students. I would not be in a position to accept any of this help and support, however, without the love of my family. Thanks, mom and dad, for nurturing my love for math from my childhood. Finally, I want to thank my wife, Hannah Lennett, who lovingly supported me through this entire process, while also picking up a graduate degree from UC Berkeley herself.

I end this section with the wise words of Charlie Alon: "It's just math."

Chapter 1

Introduction

The Tacoma Narrows Bridge failed catastrophically on November 7th, 1940. The initial report [1] noted that the oscillations were caused by the interaction of the wind with the structure of the suspension bridge that had opened just 129 days earlier. This was an instance of Aeroelastic flutter, where fluid forces and natural vibrational modes of a structure form a positive feedback loop, leading to large, and possibly damaging, oscillations.

The study of fluid-structure interaction has applications well beyond computational testing in civil engineering of bridges, building exteriors, nuclear power plants [77], and other stationary but flexible structures whose structural integrity is important. Modelling FSI is key for the aerospace industry [50], where weight savings are important, but unmitigated aeroelastic flutter can be catastrophic. Noise is another concern addressed by FSI simulations. Engine vibrations in vehicles can be modelled [66] to anticipate and reduce noise in automotive applications. Modeling the noise reduction properties of interior walls in buildings [2] is another application of FSI simulations, whether this be in an apartment building, where absorption is critical, or a concert hall, where a balance must be struck to optimize the interior acoustics.

The development of wind turbines [45] is another key application of FSI models. The newest designs are very large, with blades approaching 100 meters in length, and outputs approaching ten megawatts each. For machines of this size, full-scale physical modelling is not feasible. In comparing and evaluating designs, engineers want to maximize the amount of energy harvested from the wind at a wide range of wind speeds. Energy dissipation due to high frequency oscillations of the blades is not desirable. FSI modelling is therefore critical for the building of large-scale wind farms.

In addition to efficiency concerns, the anticipation of material fatigue in wind turbines [9] is another use case for FSI modelling. Wind turbines make hundreds of millions of rotations during their lifetimes. They are usually made of lightweight, strong composite material, but this material does degrade over time. Sensors are installed in some wind turbine blades to monitor this, but they do not directly monitor damage, relying instead on proxy measurements. These measurements are taken at a limited number of points generally far smaller than the number of points used for modelling. Thus FSI simulations are very useful in understanding the lifecycle characteristics of large-scale wind installations.

Another application is in the study of biology. One particular example is the modeling of a bat wing [5], where bones are modeled as rigid structures connected by a membrane. In this case, the lift and drag may be calculated for a particular flapping motion, leading to a better understanding of the physics of the animal. Another relevant example from the field of medicine is in the modelling of blood vessels [13] in individual patients. As fluid flows through arteries, the walls stretch in response to the increased pressure. With modern MRI and CT scans, we have non-invasive access to the blood vessel configurations of living patients at pressure. With FSI modelling, it is possible to calculate the resting configuration of parts of the circulatory system, leading to better understanding and treatment of cardiovascular disease.

1.1 Previous Work

Many numerical methods have been proposed for the time-integration of problems with coupled fluid-structure interaction. Some, such as the approaches presented in [40] and [25], use Explicit Runge-Kutta (ERK) methods, though implicit methods are more commonly used due to their improved stability properties. Some FSI solvers apply multi-step methods, such as Backward Differentiation Formulae, for time integration [75, 73, 47], but other methods have also been considered [70, 26, 74, 49]. Another option is to treat time as an extra dimension and apply space-time methods [6, 67, 34], although these typically increase the computational cost compared to standard time-integrators. In this work, we use one-step Implicit Runge-Kutta (IRK) methods because of their excellent stability properties and straight-forward extension to arbitrarily high orders of convergence.

One popular class of methods for FSI time integration are based on the so-called partitioned approach. This differs from the monolithic approach, which combines the FSI problem into one single system, increasing stability and facilitating the direct use of IRK methods or other standard high-order time-integration techniques, as shown in [24, 35, 52]. However, the partitioned approach allows for reusing existing solvers for the fluid and the structure, separately, while the time-integration scheme is chosen such that the coupling between them is properly accounted for. These partitioned methods have clear advantages, mainly the separation between the two physical systems and the ability to use separate schemes to integrate them with readily available solvers, as in [48]. Our approach to partitioned time-integration of fluid-structure interaction problems is closely related to *operator splitting* or *fractional step methods*, examples of which can be found in [39, 15], and discussed further in [23]. Some of the drawbacks with partitioned schemes can be overcome with subiterations at each timestep, as in [51, 76, 20, 48, 36], leading to the monolithic solution. These methods can be applied to more general multiphysics problems, as in [65], and not just fluid-structure interaction simulations. The method presented in this work lends itself quite readily to subiterations, though we do not find them necessary in every application studied, and, in general, forgo their use and the associated cost.

Many partitioned time-integration schemes are based on some sort of predictor-corrector framework [29, 11], often only achieving first [61, 28, 14, 53] or second [32, 27, 7, 8, 55, 16] orders of accuracy. In [78, 71, 72], a class of high-order schemes were derived based on implicit-explicit (IMEX) Runge-Kutta schemes, which allow for the computation of predicted tractions that yield higher-order accuracy. These techniques were further developed and used in the setting of discontinuous Galerkin simulations in [30, 54], and extended to a broader class of problems and solvers in [37].

The implicit time-integrators in these IMEX schemes are based on Diagonally Implicit Runge-Kutta (DIRK) methods. These are convenient since they split each stage into a single non-linear system to solve, similar to the (low-order) backward Euler method. However, this limits the accuracy and the stability of the resulting scheme, which has led to an interest in (fully) Implicit Runge-Kutta (IRK) schemes, as used in the monolithic approach in [24], and, with first-order coupling, in [69]. These IRK schemes have many excellent properties, such as higher stage-order and leading-order error coefficients which are typically smaller than those found in the DIRK methods to which we compare them. However, they couple all the stages in the Runge-Kutta scheme, which means they either require more degrees of freedom or they have to be solved using specialized solvers.

In [56], a new iterative technique was proposed for solving the systems that arise from (single-physics) IRK schemes. The method was based on a change of variable which increased the sparsity, and a scaled block-Jacobi solver which allowed for individual solution of each stage as part of the iterative scheme. These could then be solved using any existing solver, such as an ordered ILU preconditioner, much like the case for the DIRK schemes. Another benefit was the ability to parallelize across the stages, which allowed for further performance improvements. The resulting scheme gave superior results to the standard DIRK schemes, when comparing work vs. accuracy.

1.2 Overview

In this work, we present a partitioned approach for IRK integration of FSI systems, based on the iterative solvers in [56] together with various predictors for the traction. We consider both explicit and implicit predictors, and observe that while the implicit predictors are more expensive, they can provide superior stability. We demonstrate the methods on several model problems. The resulting schemes have comparable or better performance than the partitioned IMEX solvers in [78, 30], while giving the additional advantages of a full IRK scheme.

This dissertation is organized as follows. First we present the governing equations of the fluids we model, both viscous and non-viscous. This is followed by a discussion of the way in which we modify these equations to handle a deforming domain. We then lay out the governing equations of the structures we model. These consist of rigid bodies with simple governing equations, and the more complex flexible structure we model.

Having laid out the governing equations for structure and fluid individually, we then describe in detail the Runge-Kutta methods we use to move our simulations forward in time. We discuss the relatively simple ERK methods, and classify various types of more complicated implicit methods. After laying out the order conditions and stability properties of such methods, we explain the standard blended implicit-explicit Runge-Kutta formulation. Finally, we develop a novel type of implicit-explicit Runge-Kutta method which integrates forward to multiple points in time, and is used later in the work as a basis for our implicit prediction methods.

We then present standard discontinuous Galerkin methods for fluids and discontinuous Galerkin methods for structures, as well as the solvers we use to apply Runge-Kutta methods to these discretizations. This is followed by a discussion of the methods whereby we model the interaction of fluids and structures. These elements are then combined to fully describe the methods we use, both the DIRK method previously developed in [30], and the novel fully-implicit Runge-Kutta FSI method we present here. This is followed by validation of the method and applications to model problems.

Chapter 2

Governing Equations

2.1 Compressible Navier-Stokes

In general, we model the fluid using the compressible Navier-Stokes equations,

$$\frac{\partial}{\partial t}(\rho) + \frac{\partial}{\partial x_j}(\rho u_j) = 0 \tag{2.1}$$

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}(\rho u_i u_j + p\delta_{ij}) = \frac{\partial}{\partial x_j}\tau_{ij} \tag{2.2}$$

$$\frac{\partial}{\partial t}(\rho E) + \frac{\partial}{\partial x_j}(\rho u_j E + u_j p) = \frac{\partial}{\partial x_j}\left(-q_j + u_i \tau_{ij}\right) \tag{2.3}$$

as written in conservation form using Einstein indicial notation, with i from 1 to D, the number of dimensions. These are written in the conserved variables fluid density ρ, momentum in the direction of the j^{th} spatial dimension ρu_j, and total energy ρE. The variable τ_{ij} denotes the viscous stress tensor. It is defined by

$$\tau_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}\right) \tag{2.4}$$

and the heat flux q_j is defined by

$$q_j = -\frac{\mu}{\mathrm{Pr}}\frac{\partial}{\partial x_j}\left(E + \frac{p}{\rho} - \frac{1}{2}u_k u_k\right) \tag{2.5}$$

where μ is the dynamic viscosity, and Pr is the Prandtl number. In our applications, we choose $\mu = 1.83 \times 10^{-5}\,\frac{\mathrm{kg}}{\mathrm{m\cdot s}}$, and $\mathrm{Pr} = 0.72$, as these are the values for air at the standard temperature and pressure of 273.15 K and 1 bar. Modelling the fluid as an ideal gas, the pressure p is given by

$$p = (\gamma - 1)\left(\rho E - \frac{1}{2}\rho u_k u_k\right) \tag{2.6}$$

which is derived from the ideal gas law $p = \rho R T$ and the relationships $R = C_P - C_V$, $E = C_v T + \frac{1}{2} u_k u_k$, and $\gamma = \frac{C_P}{C_V}$ where C_P is the heat capacity of the fluid at a constant pressure, and C_V is the heat capacity of the fluid at a constant volume. Their ratio, γ, is the adiabatic gas constant, which we set to $\gamma = 1.4$. This is the theoretical value for a diatomic ideal gas such as O_2 or N_2, which together make up about 99% of the Earth's atmosphere at sea level.

We generally write the Navier-Stokes equations, 2.1, 2.2, & 2.3, with vector notation, writing

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}^{\mathrm{i}}(\boldsymbol{u}) - \nabla \cdot \boldsymbol{F}^{\mathrm{v}}(\boldsymbol{u}, \nabla \boldsymbol{u}) = 0 \tag{2.7}$$

with conserved variables and inviscid and viscous flux in the k^{th} spatial direction written as

$$\boldsymbol{u} = \begin{bmatrix} \rho \\ \rho u_j \\ \rho E \end{bmatrix}, \; \boldsymbol{F}^{\mathrm{i}}_k = \begin{bmatrix} \rho u_k \\ \rho u_j u_k + \delta_{jk} p \\ u_k(\rho E + p) \end{bmatrix}, \text{ and } \boldsymbol{F}^{\mathrm{v}}_k = \begin{bmatrix} 0 \\ \tau_{jk} \\ -q_j - u_i \tau_{jk} \end{bmatrix}. \tag{2.8}$$

Note that the superscript i denotes "inviscid" and is not an index in this notation.

The enthalpy in this case is $H = E + \frac{p}{\rho}$ and the speed of sound is $a = \sqrt{\frac{\gamma p}{\rho}}$. We visualize numerical solutions to these equations by plotting vorticity, $\omega = \nabla \times u_i$, entropy, $s = \frac{p}{\rho^\gamma}$, or Mach number $M = \frac{\sqrt{u_k u_k}}{a} = \sqrt{\frac{\rho u_k u_k}{\gamma p}}$. In two dimensions, vorticity can be plotted as a scalar, as its value has only one component. We also note that the internal energy, e, is defined as

$$e = E - \frac{u_k u_k}{2}. \tag{2.9}$$

This quantity is important for setting boundary conditions, and is proportional to temperature by the heat capacity at constant volume. This implies

$$e = C_V T. \tag{2.10}$$

One way to reduce the number of degrees of freedom from the system is make the assumption that the entropy s is constant. This implies that, in an ideal gas,

$$p = s \rho^\gamma, \tag{2.11}$$

which allows us to eliminate the energy equation, equation 2.3.

2.1.1 Euler Equations

In addition to modeling viscous flow, we also model inviscid flow using the Euler equations. These are another simplification of the compressible Navier-Stokes equations, where the dynamic viscosity is set to zero. It is clear from equations 2.4 and 2.5 that $\mu = 0$ implies $\tau_{ij} = q_j = 0$, and thus $\boldsymbol{F}^{\mathrm{v}} = 0$. This leaves

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}^{\mathrm{i}}(\boldsymbol{u}) = 0, \tag{2.12}$$

again with pressure defined as in 2.6. In one dimension, we drop the dimension subscripts and get the equation

$$\frac{\partial}{\partial t}\begin{bmatrix} \rho \\ \rho u \\ \rho E \end{bmatrix} + \frac{\partial}{\partial x}\begin{bmatrix} \rho u \\ \rho u^2 + p \\ u\left(\rho E + p\right) \end{bmatrix} = 0 \tag{2.13}$$

where x is the spatial coordinate.

2.1.2 Boundary Conditions

In fluid-structure interaction simulations, it is necessary to impose two different boundary conditions. The first is imposed at a interfaces of the fluid with a solid, either a dynamic structure or a stationary edge. Defining $\vec{v}_{\text{fluid}}$ to be the velocity of the fluid, and $\vec{v}_{\text{wall}}$ to be the velocity of the structure or stationary edge (generally zero), we impose *adiabatic no-slip* boundary conditions, where

$$\vec{v}_{\text{fluid}} = \vec{v}_{\text{wall}} \tag{2.14}$$

and no heat or mass passes through the interface.

The second boundary condition is imposed at the outer boundary of the fluid domain, away from the structure. Since there is a limit on computational work and memory storage, the computational domain must be finite. We impose *far-field* boundary conditions at the edge of the computational domain, with sufficient distance from the structure so as to avoid any significant effects of the fluid-structure interaction. Far-field boundary conditions are imposed by directly imposing the state of the system, with state $\boldsymbol{u}_\infty$, defined with the far-field state imposed as

$$\boldsymbol{u}_\infty = \begin{bmatrix} \rho_\infty \\ \rho_\infty \vec{v}_\infty \\ \frac{1}{\gamma-1}p_\infty + \frac{1}{2}\rho_\infty \|\vec{v}_\infty\|_2^2 \end{bmatrix} \tag{2.15}$$

with the pressure p_∞, density ρ_∞, and velocity $\vec{v}_\infty$ chosen to match their values beyond the computational domain.

2.2 Arbitrary Lagrangian Eulerian Formulation

As the structure moves, it displaces the fluid, changing its shape. Because of this effect, we employ an Arbitrary Lagrangian Eulerian (ALE) framework to solve the conservation laws presented in 2.1 on a fixed reference domain. Whereas an Eulerian formulation is fixed in space, with fluid moving through elements, and a Lagrangian formulation has elements fixed on a moving material, an ALE formulation is neither fixed to the material nor fixed in space, but offers a change of variables conversion from the system of conservation laws on our varying physical domain into a system of conservation laws in the fixed reference domain. We present this ALE formulation using the standard notation of capital letters for the variables associated with the static reference domain V, and lowercase letters for those of the deforming physical domain $v(t)$.

Consider a point $\boldsymbol{X} \in V$ which is mapped at time t to $\boldsymbol{x}(\boldsymbol{X}, t) \in v(t)$. We define the deformation gradient $\boldsymbol{G}$ at $\boldsymbol{x}$ as

$$\boldsymbol{G} = \nabla_{\boldsymbol{X}} \boldsymbol{x} \tag{2.16}$$

which describes how space is deformed at the point x. The local volume deformation is thus

$$g = \det \boldsymbol{G} \tag{2.17}$$

with $g = 1$ denoting no volume deformation. The mapping velocity is defined as

$$\dot{\boldsymbol{x}} = \frac{\partial \boldsymbol{x}}{\partial t}. \tag{2.18}$$

The relationships between the normal vector $\boldsymbol{n}$, elemental area da, and elemental volume dv in the physical coordinates and normal vector $\boldsymbol{N}$, elemental area dA, and elemental volume dV in the reference coordinates is derived in [60]. We state them here for brevity:

$$\boldsymbol{n}\ da = g \boldsymbol{G}^{-T} \boldsymbol{N}\ dA \tag{2.19}$$

$$\boldsymbol{N}\ dA = g^{-1} \boldsymbol{G}^{T} \boldsymbol{n}\ da \tag{2.20}$$

$$dv = g\ dV \tag{2.21}$$

Applying this to a system of conservation laws

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{f}(\boldsymbol{u}, \nabla \boldsymbol{u}) = 0 \tag{2.22}$$

and integrating over the physical domain $v(t)$ we get

$$\int_{v(t)} \frac{\partial \boldsymbol{u}}{\partial t}\ dv + \int_{v(t)} \nabla \cdot \boldsymbol{f}(\boldsymbol{u}, \nabla \boldsymbol{u})\ dv = 0 \tag{2.23}$$

Application of the divergence theorem results in the integral form of the system

$$\int_{v(t)} \frac{\partial \boldsymbol{u}}{\partial t}\ dv + \int_{\partial v(t)} \boldsymbol{f}(\boldsymbol{u}, \nabla \boldsymbol{u}) \cdot \boldsymbol{n}\ da = 0. \tag{2.24}$$

We apply Reynolds transport theorem to the first term to get

$$\frac{d}{dt} \int_{v(t)} \boldsymbol{u}\ dv - \int_{\partial v(t)} (\dot{\boldsymbol{x}} \cdot \boldsymbol{n}) \boldsymbol{u}\ da + \int_{\partial v(t)} \boldsymbol{f}(\boldsymbol{u}, \nabla \boldsymbol{u}) \cdot \boldsymbol{n}\ da = 0. \tag{2.25}$$

which is still in the physical domain. To change to the reference domain, we change coordinates in accordance with equations 2.19, 2.20, and 2.21, and apply the Reynolds transport theorem yet again to get the equation

$$\int_V \frac{\partial(g^{-1}\boldsymbol{u})}{\partial t}\, dV - \int_{\partial V} \left(g\boldsymbol{u}\boldsymbol{G}^{-1}\dot{\boldsymbol{x}}\right)\cdot \boldsymbol{N}\, dA + \int_{\partial V} \left(g\boldsymbol{G}^{-1}\boldsymbol{f}\right)\cdot \boldsymbol{N}\, dA = 0 \tag{2.26}$$

which is in the reference domain. Applying the divergence theorem and localization theorem once again removes the integrals to give the equation

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla_{\boldsymbol{X}}\cdot \boldsymbol{F}(\boldsymbol{U}, \nabla_{\boldsymbol{X}}\boldsymbol{U}) = 0 \tag{2.27}$$

where

$$\boldsymbol{U} = g\boldsymbol{u} \text{ and } \boldsymbol{F} = g\boldsymbol{G}^{-1}\boldsymbol{f} - \boldsymbol{U}\boldsymbol{G}^{-1}\dot{\boldsymbol{x}} \tag{2.28}$$

are the conserved quantities and flux, respectively, in the reference domain. Applying the chain rule to these definitions, we obtain the gradient

$$\nabla\boldsymbol{u} = (\nabla_{\boldsymbol{X}}(g^{-1}\boldsymbol{U})\boldsymbol{G}^{-T} = \left(g^{-1}\nabla_{\boldsymbol{X}}\boldsymbol{U} - \boldsymbol{U}\nabla_{\boldsymbol{X}}(g^{-1})\right)\boldsymbol{G}^{-T}. \tag{2.29}$$

This gives a complete picture of the conservation laws we seek to solve on the reference domain of the ALE formulation.

Next we split the flux function into two parts, as in equation 2.7. This involves ascribing the term $\boldsymbol{U}\boldsymbol{G}^{-1}\dot{\boldsymbol{x}}$ to either the inviscid or viscous term. We ascribe it to the inviscid term and get

$$F^{\mathrm{i}} = g\boldsymbol{G}^{-1}\boldsymbol{f}^{\mathrm{i}} - \boldsymbol{U}\boldsymbol{G}^{-1}\dot{\boldsymbol{x}} \tag{2.30}$$

$$F^{\mathrm{v}} = g\boldsymbol{G}^{-1}\boldsymbol{f}^{\mathrm{v}}. \tag{2.31}$$

In general, though constant solutions in the physical domain $v(t)$ should be solutions of the discretized equations in the reference domain V, under the ALE formulation just described this is not guaranteed to be the case. This can be rectified by enforcing the geometric conservation law (GCL) described in [68] by replacing the equation 2.27 by

$$\frac{\partial \bar{g}g^{-1}\boldsymbol{U}}{\partial t} + \nabla_{\boldsymbol{X}}\cdot \boldsymbol{F} = 0 \tag{2.32}$$

where the quantity $\bar{g}$ is derived by integrating the auxiliary equation

$$\frac{\partial \bar{g}}{\partial t} = \nabla_{\boldsymbol{X}}\left(g\boldsymbol{G}^{-1}\dot{\boldsymbol{x}}\right). \tag{2.33}$$

For simplicity, however, We do not make this replacement. Though the errors resulting from violating the GCL generally relatively large for low-order methods, the errors can be somewhat smaller in higher-order methods. In our applications, we have not found them to be a problem. For more information, see [60].

2.3 Rigid Body Dynamics

The equations of motion for a rigid body of mass m and center of mass $\boldsymbol{x}_m$ can be derived from a simple application of Newton's second law, that the change in momentum over time is exactly equal to the force applied. Since our mass is invariant,

$$\boldsymbol{F} = m\boldsymbol{a} \tag{2.34}$$

where $\boldsymbol{F}$ is the sum of the forces acting on the body, and

$$\boldsymbol{a} = \frac{d^2\boldsymbol{x}_m}{dt^2}. \tag{2.35}$$

In general, this force is applied by the fluid as a surface traction $\boldsymbol{t}$, in force per unit area. Integrating over the whole surface,

$$\boldsymbol{F} = \int_{\partial V} \boldsymbol{t}\, dA \tag{2.36}$$

gives the net force on the body.

As the state of the rigid body is uniquely determined by the position of the center of mass and the orientation, all that remains is the set of equations governing the Euler angles, which define the orientation. This follows from the law

$$\boldsymbol{\tau} = I\boldsymbol{\alpha} \tag{2.37}$$

where $\boldsymbol{\tau}$ is the net torque acting at $\boldsymbol{x}_m$, I is the moment of inertia tensor, and $\boldsymbol{\alpha}$ is the angular acceleration, defined as

$$\boldsymbol{\alpha} = \frac{d^2\boldsymbol{\theta}}{dt^2} \tag{2.38}$$

where $\boldsymbol{\theta}$ is the vector of Euler angles. If the torque is applied by surface traction, this can be calculated by the integral

$$\boldsymbol{\tau} = \int_{\partial V} (\boldsymbol{x} - \boldsymbol{x}_m) \times \boldsymbol{t}\, dA \tag{2.39}$$

over the surface area.

Though we present the full three-dimensional equations for rigid body motion here, our applications are limited to two dimensions. This means that the axis of rotation is fixed, and θ, α, and τ can effectively be treated as scalars. In the case of linear force, the integral in equation 2.36 reduces to one dimension. In one dimension, no rotation is possible, and the motion of the body is determined solely by equation 2.34, with the force equal to the traction at the interface point.

2.4 Neo-Hookean Elasticity Model

The governing equations are somewhat more complicated for flexible structures than rigid bodies. We model deformable structures V using a hyperelastic neo-Hookean formulation, found in [30]. As with the notation in section 2.2, we use lowercase letters for the physical domain and capital letters for the undeformed reference domain, mapping $\boldsymbol{x}(\boldsymbol{X}, t)$ to $\boldsymbol{X}$ at time t. As in the ALE formulation, the mapping velocity is important, and is defined as

$$\boldsymbol{v} = \frac{\partial \boldsymbol{x}}{\partial t} \tag{2.40}$$

while the deformation gradient is given by

$$\boldsymbol{F} = \nabla_{\boldsymbol{X}} \boldsymbol{x}(\boldsymbol{X}, t) \tag{2.41}$$

and the determinant of the deformation gradient is defined as

$$J = \det \boldsymbol{F}. \tag{2.42}$$

The first invariant of the deviatoric part of the left Cauchy-Green deformation tensor is given by

$$\bar{I}_1 = J^{-\frac{2}{3}} \mathrm{tr}\left(\boldsymbol{F}\boldsymbol{F}^T\right) \tag{2.43}$$

We wish to fix the structure in place on some portion of the boundary of V, and allow others to move under the influence of a traction $\boldsymbol{t}$. This is done by assigning the fixed parts of the boundary ∂V Dirichlet boundary conditions, while the free-moving parts are assigned Neumann boundary conditions. These are denoted Γ_D and Γ_N, respectively. The fixed motion of Γ_D is described by $\boldsymbol{x}_D$. The traction $\boldsymbol{t}$ is a force per unit surface area. The governing equation for the flexible structure in the interior V is

$$\frac{\partial \boldsymbol{p}}{\partial t} - \nabla \cdot \boldsymbol{P}(\boldsymbol{F}) = \boldsymbol{b} \tag{2.44}$$

while the equation on Γ_N is

$$\boldsymbol{P}(\boldsymbol{F}) \cdot \boldsymbol{N} = \boldsymbol{t} \tag{2.45}$$

and on Γ_D

$$\boldsymbol{x} = \boldsymbol{x}_D \tag{2.46}$$

where

$$\boldsymbol{p} = \rho \boldsymbol{v} = \rho \frac{\partial \boldsymbol{x}}{\partial t} \tag{2.47}$$

is the momentum of the structure, $\boldsymbol{b}$ is an external body force per unit reference volume (usually zero), and $\boldsymbol{N}$ is a unit normal vector in the reference domain. The first Piola-Kirchhoff stress tensor, $\boldsymbol{P}$ is defined as

$$\boldsymbol{P}(\boldsymbol{F}) = \frac{\partial W}{\partial \boldsymbol{F}} \tag{2.48}$$

where W is the strain energy density of material, which depends on the shear and bulk moduli μ and κ by the equation

$$W = \frac{\mu}{2}\left(\bar{I}_1 - 2\right) + \frac{\kappa}{2}\left(J - 1\right)^2. \tag{2.49}$$

As we are working in two dimensions, we treat the stretch in the third dimension as constant. ($\bar{I}_1 - 2$ becomes $\bar{I}_1 - 3$ in three dimensions.) This means that the Piola-Kirchhoff stress tensor is

$$\boldsymbol{P}\left(\boldsymbol{F}\right) = \mu J^{-\frac{2}{3}}\left(\boldsymbol{F} - \frac{1}{3}\left(\operatorname{tr}\left(\boldsymbol{F}\boldsymbol{F}^T\right) + 1\right)\boldsymbol{F}^{-T}\right) + \kappa\left(J - 1\right)J\boldsymbol{F}^{-T}. \tag{2.50}$$

As this does not go to infinity as J goes to zero, this model does not satisfy the requirements of Ball's existence theorem for finite elasticity. In practice, however, this is not a problem in the applications we consider in this document.

It should be noted that we generally do not describe our parameters in terms of shear and bulk modulus, but in terms of Young's modulus E and Poisson's ratio ν. These are

$$\mu = \frac{E}{2\left(1 + \nu\right)} \quad \text{and} \quad \kappa = \frac{E}{3\left(1 - 2\nu\right)}. \tag{2.51}$$

Chapter 3

Runge-Kutta Temporal Discretization

We evolve the partial differential equations described in chapter 2 in time using the *method of lines*. This is done by discretizing the equations in space, as described in chapter 4, then integrating the resulting vector-valued ordinary differential equation forward in time using Runge-Kutta methods. The ODE developed in chapter 4 is

$$M\frac{du}{dt} = R(t, u) \tag{3.1}$$

with initial conditions

$$u(t_0) = u_0. \tag{3.2}$$

Using the Galerkin methods we employ results in a non-singular matrix M and a Lipschitz function R in equation 3.1, allowing us to apply both explicit and implicit Runge-Kutta methods to the ODE to integrate one timestep Δt from t_n to $t_{n+1} = t_n + \Delta t$, calculating u_{n+1} from u_n.

3.1 Explicit Runge-Kutta Methods

The most basic explicit Runge-Kutta (ERK) method is *forward Euler*, which is derived by setting the solution at time t_{n+1}, called u_{n+1}, equal to the first two terms of the Taylor expansion of R about t_n:

$$Mu_{n+1} = Mu_n + \Delta t R(t_n, u_n) \tag{3.3}$$

This method can be implemented by evaluating $b = R(t_n, u_n)$ and solving the linear equation $Mk = b$ for k, essentially solving

$$Mk = R(t_n, u_n). \tag{3.4}$$

To finish the implementation of forward Euler's method, we set

$$u_{n+1} = u_n + \Delta t k. \tag{3.5}$$

Forward Euler has been improved upon significantly, beginning with [64]. (A history of Runge-Kutta is found in [18].) One example is *Heun's method*, a slightly more complicated Runge-Kutta method which consists of solving

$$Mk_1 = R(t_n, u_n) \tag{3.6}$$

$$Mk_2 = R(t_n + \Delta t, u_n + \Delta t k_1) \tag{3.7}$$

for two *stages* k_1 and k_2, and advancing u_n by

$$u_{n+1} = u_n + \Delta t \left(\frac{1}{2} k_1 + \frac{1}{2} k_2 \right). \tag{3.8}$$

Analysis of the Taylor expansion of R about u_n and $u_n + \Delta t k_1$ reveals that the error at each step is reduced by using Heun's method. This per-step error can be further reduced by replacing equation 3.7 with

$$Mk_2 = R\left(t_n + \frac{2}{3}\Delta t, u_n + \Delta t \frac{2}{3} k_1 \right) \tag{3.9}$$

and advancing u_n by

$$u_{n+1} = u_n + \Delta t \left(\frac{1}{4} k_1 + \frac{3}{4} k_2 \right) \tag{3.10}$$

in a Runge-Kutta method known as *Ralston's Second-Order Method*. These methods can be interpreted as using evaluations of R at t_n and $t_n + c\Delta t$ in linear combination to advance the solution u_n to u_{n+1}, with $c = 1$ for Heun's method, and $c = \frac{2}{3}$ for Ralston's second-order method. Note that at each stage, R must be evaluated and one linear solve of the linear system $Mk = b$, meaning that the two stage methods are twice as expensive per timestep as forward Euler's method, though, because we will solve these systems in a static reference domain, M remains unchanged, and this cost can be quite low.

A general Runge-Kutta method of s stages is written as

$$Mk_i = R\left(t_n + c_i \Delta t, u_n + \Delta t w_i\right) \tag{3.11}$$

$$u_{n+1} = u_n + \Delta t \sum_{i=1}^{s} b_i k_i \tag{3.12}$$

where a_{ij} are called coefficients, b_i weights, and c_i nodes, and we define w_i throughout this document as

$$w_i = \sum_{j=1}^{s} a_{ij} k_j \tag{3.13}$$

Runge-Kutta methods are usually presented as a Butcher tableau:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \tag{3.14}$$

In this form, forward Euler, Heun's method, and Ralston's second-order methods are expressed as

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \quad \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \quad \begin{array}{c|cc} 0 & 0 & 0 \\ \frac{2}{3} & \frac{2}{3} & 0 \\ \hline & \frac{1}{4} & \frac{3}{4} \end{array} \tag{3.15}$$

Generally, however, zeros in the tableau are left implied. For example, the Kutta's third-order method and the classical Runge-Kutta method from [46], RK4, are

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \quad \begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & & \frac{1}{2} & & \\ 1 & & & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}. \tag{3.16}$$

It should be noted that explicit Runge-Kutta methods have the property that w_i depends only on the preceding stages. This implies that the A matrix in the Butcher tableau is strictly lower-triangular, and the stages can be calculated in sequence explicitly, solving a linear equation of the form 3.11 explicitly at each stage.

3.2 Implicit Runge-Kutta Methods

Implicit Runge-Kutta methods, described in [17], are clearly identifiable by their butcher arrays. A Runge-Kutta method whose Butcher tableau is lower-triangular with non-zero entries on the diagonal is a *Diagonally Implicit Runge-Kutta Method* (DIRK). Each stage can be calculated in sequence, though at each stage i with a non-zero entry in the diagonal equation k_i is implicitly defined by equation 3.11 as w_i is dependent on k_i itself. This leads to our approximating a zero of

$$F(k_i) = Mk_i - R(t_n + c_i\Delta t, u_n + \Delta t w_i). \tag{3.17}$$

We use Newton's method for this, beginning with a guess (usually zero, for simplicity), and solving the linear equation

$$(M - \Delta t a_{ii} J_R(t_n + c_i\Delta t, u_n + \Delta t w_i))\,\Delta k_i = R(t_n + c_i\Delta t, u_n + \Delta t w_i) - Mk_i \tag{3.18}$$

with a linear solver, updating k_i by adding Δk_i at each iteration, until tolerance is achieved. Note that J_R is the Jacobian matrix of the residual R. Though this is more expensive than solving equation 3.11 in the explicit case, DIRK methods have stability advantages discussed further below.

Two special cases of DIRK methods are *singly diagonally implicit* (SDIRK) methods, where all diagonal entries of A are equal, and *explicit first stage, singly diagonally implicit* (ESDIRK) methods, where all diagonal entries of A are equal, except for a_{11}, which is set to zero. SDIRK methods offer the advantage that the Jacobian matrix in equation 3.18 is scaled by the same constant at each stage, simplifying implementation. ESDIRK methods have the advantage of offering an easily calculated first stage k_1 before more stages are calculated implicitly.

A Runge-Kutta method whose Butcher tableau is dense is called an *Implicit Runge-Kutta* method (IRK) or *Fully Implicit Runge-Kutta* method. All stage values k_i are interdependent, prohibiting their calculation sequentially. The complete system is

$$(\boldsymbol{I_s} \otimes M)\boldsymbol{K} = \boldsymbol{R}\left(t_0 + \Delta t\boldsymbol{c}, \boldsymbol{U}_n + \Delta t(\boldsymbol{A} \otimes I_n)\boldsymbol{K}\right), \tag{3.19}$$

where $\boldsymbol{K}$, $\boldsymbol{R}$ and $\boldsymbol{U}_n$ are concatenations of k_i, f and u_n, respectively (using boldface for multi-stage objects). When A is invertable,

$$\boldsymbol{W} = (\boldsymbol{A} \otimes I_n)\boldsymbol{K} \tag{3.20}$$

where $\mathbf{W}$ is the concatenation of of w_i, becomes

$$\boldsymbol{K} = (\boldsymbol{A}^{-1} \otimes I_n)\boldsymbol{W} \tag{3.21}$$

and this allows the transformation of equation 3.19 into

$$(\boldsymbol{A}^{-1} \otimes M)\boldsymbol{W} = \boldsymbol{R}(t_0 + \Delta t\boldsymbol{c}, \boldsymbol{U}_0 + \Delta t\boldsymbol{W}). \tag{3.22}$$

The time advancement in equation 3.12 can be written in the transformed variables as

$$u_{n+1} = u_n + \Delta t(\boldsymbol{b}^T\boldsymbol{A}^{-1} \otimes I_n)\boldsymbol{W}. \tag{3.23}$$

Analogous to what is done with equation 3.18, we begin with a guess for $\mathbf{W}$, then solve the linear equation from Newton's method

$$\left(\boldsymbol{A}^{-1} \otimes M - \Delta t\boldsymbol{J}(t_0 + \Delta t\boldsymbol{c}, \boldsymbol{U}_0 + \Delta t\boldsymbol{W})\right)\Delta\boldsymbol{W} = \boldsymbol{R}(t_0 + \Delta t\boldsymbol{c}, \boldsymbol{U}_0 + \Delta t\boldsymbol{W}) - \left(\boldsymbol{A}^{-1} \otimes M\right)\mathbf{W} \tag{3.24}$$

and update $\mathbf{W}$ by adding $\Delta\mathbf{W}$ at each iteration until tolerance is reached. This creates a linear system of size s times larger than with DIRK methods, but leads to certain advantages discussed below. A summary of each type of Runge-Kutta discussed here can be found in table 3.1.

These classes of Runge-Kutta methods can be further characterised by their stage order, defined for a p^{th}-order Runge-Kutta method as $q = \min\{p, q_i\}$, where q_i is defined as

$$u(t_n + \Delta t c_i) = u_n + \Delta t w_i + \mathcal{O}\left(\Delta t^{q_i+1}\right) \tag{3.25}$$

for all i from 1 to s. The maximum stage order for any DIRK method is $q = 2$, achieved only in EDIRK and ESDIRK methods, whereas ERK and DIRK methods have maximum stage order $q = 1$. Furthermore, though ERK methods have the attractive property of involving no non-linear solves, they lack the stability achievable with other Runge-Kutta methods. As discussed in [56], high stage order and stability are particularly important for stiff problems. Though DIRK, EDIRK, ESDIRK, and IRK methods can all achieve L-stability, DIRK, EDIRK, and ESDIRK methods do so with relatively small non-linear solves, but a limit on stage order, while IRK methods exist with no such limitation on stage order, but requiring larger non-linear solves.

Many examples of IRK methods have been developed, but we will focus on two main types: Radau IIA and Lobatto IIIC methods. Radau IIA methods are L-stable with order $2s - 1$ and stage order s. The first three have Butcher tableaus

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \qquad \begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & \frac{-1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array} \qquad \begin{array}{c|ccc} \frac{4-\sqrt{6}}{10} & \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{4-\sqrt{6}}{10} & \frac{296-169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ 1 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \\ \hline & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array}. \tag{3.26}$$

See [56] for Radau IIA methods of higher order. Lobatto IIIC methods have order $2s - 2$ and stage order s. Like Radau IIA methods, Lobatto IIIC methods are L-stable. They are also algebraically stable and thus B-stable. The Butcher tableaus for the second-, fourth-, and sixth-order Lobatto IIIC methods are

$$\begin{array}{c|cc} 0 & \frac{1}{2} & -\frac{1}{2} \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \qquad \begin{array}{c|ccc} 0 & \frac{1}{6} & -\frac{1}{3} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{6} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \qquad \begin{array}{c|cccc} 0 & \frac{1}{12} & -\frac{\sqrt{5}}{12} & \frac{\sqrt{5}}{12} & -\frac{1}{12} \\ \frac{5-\sqrt{5}}{10} & \frac{1}{12} & \frac{1}{4} & \frac{10-7\sqrt{5}}{60} & \frac{\sqrt{5}}{60} \\ \frac{5+\sqrt{5}}{10} & \frac{1}{12} & \frac{10+7\sqrt{5}}{60} & \frac{1}{4} & -\frac{\sqrt{5}}{60} \\ 1 & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12} \\ \hline & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12} \end{array}. \tag{3.27}$$

3.3 Runge-Kutta Method Order Conditions

In addition to the classification of Runge-Kutta methods found in table 3.1, Runge-Kutta methods are also classified by their *order of accuracy*. For simplicity of notation, we focus on the one-dimensional case, but note when the analysis applies differently in the multi-dimensional case. Consider the ODE

$$y' = f(t, y) \tag{3.28}$$

Class	Property of A	System size	Systems/step
Explicit (ERK)	strictly lower-triangular	-	0
Diagonally-Implicit (DIRK)	lower-triangular	N	s
Explicit first stage Diagonally-Implicit (EDIRK)	lower-triangular & $a_{11} = 0$	N	$s - 1$
Explicit first stage Singly Diagonally-Implicit (ESDIRK)	lower-triangular, $a_{11} = 0$, & $a_{ii} = a_{jj}\ \forall\ i, j > 1$	N	$s - 1$
Implicit (IRK)	dense	$N \cdot s$	1

Table 3.1: Classes of Runge-Kutta Methods solving an ODE of N variables

where y' is the time derivative of the exact solution $y(t)$ and $f(t, y)$ is Lipschitz continuous, that is to say there exists some λ such that for any two states y and $\tilde{y}$,

$$||f(y,t) - f(\tilde{y},t)|| \leq \lambda ||y - \tilde{y}|| \tag{3.29}$$

for some norm $|| \cdot ||$. Any smoothly differentiable function satisfies this condition. Let the initial condition be $y(0) = y_0$. To define the order of a RK method, let $y_n = y(t_n)$ be the solution at time t_n, y_{n+1} be the solution at time $t_{n+1} = t_n + \Delta t$ as computed by the RK method, and $y(t_{n+1})$ be the exact solution at time t_{n+1}. (Note that in general, $y(t_{n+1}) \neq y_{n+1}$.)

A Runge-Kutta method is defined as order q (or q^{th}-order) if

$$||y(t_{n+1}) - y_{n+1}|| = \mathcal{O}(\Delta t^{q+1}) \tag{3.30}$$

which is to say that the RK method cancels out the first q terms of the Taylor series of y about t_n. This is accomplished when, adopting the notation of 3.12,

$$y_n + \sum_{i=1}^{q} \frac{\Delta t^i}{i!} y^{(i)}(t_n) + \mathcal{O}(\Delta t^{q+1}) = y_n + \Delta t \sum_{i=1}^{s} b_i k_i. \tag{3.31}$$

Consider the term $y^{(i)}(t_n)$. Dropping the function inputs for clarity, the ODE requires that, in the case of $i = 1$, $y' = f$. Applying chain rule to the $i = 2$ and $i = 3$ cases, $y'' = f_y f$ and $y''' = f_{yy} f^2 + f_y^2 f$. Setting i equal to four gives $y'''' = f_{yyy} f^3 + 4 f_{yy} f_y f^2 + f_y^3 f$. Continuing in this way, it is clear that $y^{(i)}$ is equal to a linear combination of products of y-derivatives of f. These derivatives may not be unique, leading to exponents higher than one. Each is

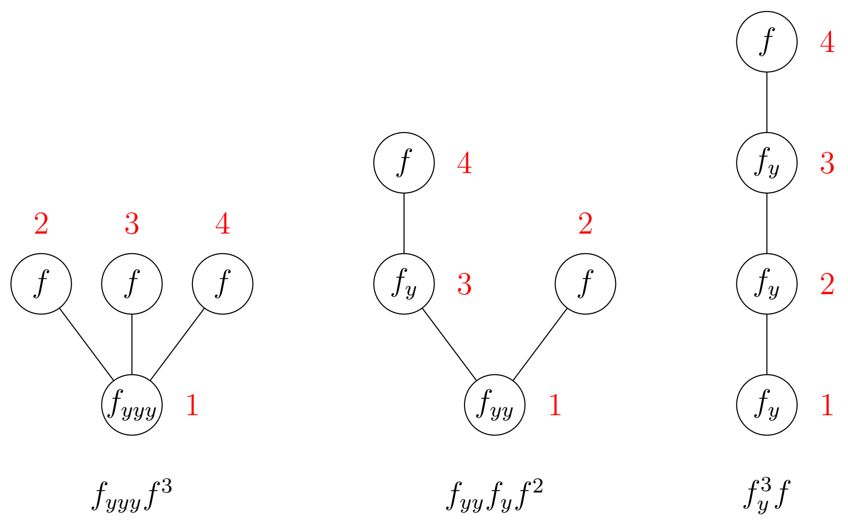


Figure 3.1: y'''' trees

associated with a rooted tree, from graph theory, as described in [22]. The connection to RK methods described below is proven in [33]. An overview of this connection is found in [19].

Consider all rooted trees $\tau_{i,j}$ with i vertices, indexed by j. When $i = 1$ or $i = 2$, $j = 1$, as there is only one such tree, having one vertex when $i = 1$, or two vertices connected when $i = 2$. When $i = 3$, j goes from one to two, as there are two trees with three vertices. For each n, we associate the n^{th} y-derivative of f with each vertex with n children. With each rooted tree $\tau_{i,j}$ we associate the product of all these y-derivatives of f from the vertices. Thereby f is associated with $\tau_{1,1}$, $f_y f$ with $\tau_{2,1}$, $f_{yy} f^2$ with $\tau_{3,1}$ (star tree), and $f_y^2 f$ with $\tau_{3,2}$ (path graph). The trees for $i = 4$ are found in figure 3.1. In this way, the linear combination equal to $y^{(i)}$ has all its components accounted for by all trees $\tau_{i,j}$, indexed by j.

Note that in the multi-dimensional case, derivatives are no longer scalars, but tensors, which are not generally commutative, prohibiting the grouping of non-unique terms. Each derivative of y is then mapped to a rooted tree labeled with additional, ordered indices such that the parent of any node comes earlier in the ordering. The requirements for each derivative of y turn out to be the same, however, so we continue with the one-dimensional case.

To describe the Runge-Kutta method order conditions, we define a function $\gamma(\tau)$ for each rooted tree τ. To define γ, we define *uprooting* of a tree as the removal of the root, and assignment of root status to all the children of the now-absent root. Uprooting results in *cuttings* of the original tree. A tree whose root has n children produces n cuttings when uprooted. $\gamma(\tau)$ is defined recursively as the product

$$\gamma(\tau_{i,j}) = i \prod_{k=1}^{n} \gamma(\tau_{i,j}^{k}) \tag{3.32}$$

where n is the number of children of the root of $\tau_{i,j}$ and each $\tau_{i,j}^{k}$ is a cutting of $\tau_{i,j}$. For the empty tree, $\gamma(\tau_0) = 1$.

To connect this to the coefficients, weights, and nodes of a RK method, consider the rooted tree $\tau_{i,j}$, which has i vertices. Choosing an arbitrary ordering for the vertices, we assign each an integer. As this is arbitrary, we assign one to the root, and the integers two through i to the other vertices. Noting that each edge connects exactly two vertices, there are $i-1$ edges, each connecting a unique pair of vertices. Thus each edge can be assigned a unique pair of non-equal integers $\{n, m\}$, with n as the integer assigned to the vertex at one end of the edge, and m the integer assigned to the vertex at the other end. We define the set of edges of a rooted tree with the function $\Psi(\tau)$. Going left to right in figure 3.1, for example, the rooted tree $\Psi(\tau_{4,1}) = \{\{1,2\},\{1,3\},\{1,4\}\}$, $\Psi(\tau_{4,2}) = \{\{1,2\},\{1,3\},\{3,4\}\}$, and $\Psi(\tau_{4,3}) = \{\{1,2\},\{2,3\},\{3,4\}\}$. From these, we define a function

$$\Phi_{k_1}(\tau) = \sum_{k_2,\cdots,k_i=1}^{s} b_{k_1} \left(\prod_{\{n,m\}\in\Psi(\tau_{ij})} a_{k_n,k_m} \right). \tag{3.33}$$

Where each k_1 is a free index and a_{k_n,k_m} are coefficients of a Runge-Kutta method with s stages. We sometimes simplify $\Phi_{k_1}(\tau)$ by imposing the consistency requirement of RK methods,

$$c_{k_n} = \sum_{k_m=1}^{s} a_{k_n,k_m} \quad \forall k_n \in \{1,\cdots,s\} \tag{3.34}$$

which replaces one of the terms a_{k_n,k_m} in the product in equation 3.33 with c_{k_n} (Specifically where $\{k_n, k_m\}$ is an edge connecting a vertex with no children).

Finally, the order condition corresponding to the rooted tree $\tau_{i,j}$ for an s-stage Runge-Kutta method is

$$\frac{1}{\gamma(\tau_{ij})} = \sum_{k_1=1}^{s} b_{k_1} \Phi(\tau_{i,j}). \tag{3.35}$$

For a RK method to be accurate to order q, it must satisfy equation 3.34 (consistency), and equation 3.35 for all rooted trees $\tau_{i,j}$ up to $i = q$.

3.4 Stability of Runge-Kutta Methods

Runge-Kutta methods can be sorted into classes based on their linear stability. This is based on each method's ability effectively solve the linear ODE $y' = \lambda y$, for a complex λ.

Since the solution to this ODE is $y(t) = y_0 e^{\lambda t}$, the solution should grow over time if the real part of λ is greater than zero, but not if the real part of λ is negative. Applying this ODE to equation 3.11,

$$k_i = \lambda \left(u_n + \Delta t \sum_{j=1}^{s} a_{ij} k_j \right) \tag{3.36}$$

and equation 3.12 becomes

$$u_{n+1} = u_n + \Delta t \sum_{i=1}^{s} \left(b_i \lambda \left(u_n + \Delta t \sum_{j=1}^{s} a_{ij} k_j \right) \right) = \left(1 + \Delta t \lambda b^T \left(I - \Delta t \lambda A \right)^{-1} e \right) u_n \tag{3.37}$$

where e is a vector of s ones. Setting $\Delta t = z$, we multiply u_n by

$$R(z) = 1 + z b^T \left(I - \Delta t \lambda A \right)^{-1} e \tag{3.38}$$

at each step. A method is called *A-stable* if $|R(z)| < 1$ for all z with negative real parts. A method is called *L-stable* if it is A-stable and

$$\lim_{z \to \infty} |R(z)| = 0 \tag{3.39}$$

A-stability and L-stability are attractive properties in a RK method, as they keep the solution u_n from "blowing up" as the timestep is increased.

In addition to linear stability, we define two types of non-linear stability here. A method is *B-stable* if for any two solutions u_n^1 and u_n^2, $||u_{n+1}^1 - u_{n+1}^2|| \leq ||u_n^1 - u_n^2||$ for all n, where u_{n+1}^i is the result of one step of an RK method applied to a dissipative ODE, beginning with u_n^i. A RK method is *algebraically stable* if the matrix $BA + A^T B^T - bb^T$ is non-negative semidefinite, where B is the s-by-s matrix with the weights of b_i on the diagonal. A RK method is B-stable if it is algebraically stable.

3.5 Implicit-Explicit Runge-Kutta Methods

The three main types of Runge-Kutta methods each have their advantages and disadvantages. ERK methods are very cheap to implement per timestep, but become unstable when applied to stiff problems unless very small timesteps are used, as they are never A-stable. DIRK methods, are more expensive to implement per timestep because of the non-linear solves involved at each stage, but can be applied to stiff problems with much larger timesteps than ERK methods, as they can be A- or L-stable, allowing for larger, and thus fewer, timesteps, incurring less cost over a finite period of time. IRK methods, though generally more expensive to implement per timestep than DIRK methods, nonetheless have even better stability properties, allowing for even larger timesteps. As discussed in 3.2, IRK methods have no inherent limit on their stage order, a particularly attractive property for exceptionally stiff problems.

In many types of problems it is advantageous to blend these different types of Runge-Kutta Methods. Consider the standard convection-diffusion equation

$$u_t = uu_x + \nu\Delta u. \tag{3.40}$$

Using a spatial discretization, as in section 4.1, we arrive at the semi-discrete ordinary differential equation

$$\frac{du}{dt} = f(u) + g(u). \tag{3.41}$$

If the equation were simply $\frac{du}{dt} = uu_x$, our ODE would be $\frac{du}{dt} = f(u)$. Employing an ERK would be the option of choice, as f is not generally stiff. We would avoid non-linear solves and not have an unreasonably small Δt, limiting the number of timesteps. If the equation were $\frac{du}{dt} = \nu\Delta u$, the resulting ODE would be $\frac{du}{dt} = g(u)$. This would be stiff, suggesting an IRK method would be most suitable. Additionally, g would be linear, meaning that the non-linear solve 3.22 would actually be a linear solve. However, the sum (equation 3.41) would require an IRK method, negating the benefits of both functions.

A common choice for a situation like this is to pair an implicit method with an explicit one, in an *implicit-explicit* (IMEX) Runge-Kutta method. As introduced in [4], we can use an ERK method with coefficients $\hat{A}$, weights $\hat{b}$, and nodes $\hat{c}$, and a DIRK method with coefficients A, weights b, and nodes c. Though not necessary, we typically require the nodes to be the same for both RK methods, so we require $c = \hat{c}$. It should be noted that for a Runge-Kutta method to be consistent,

$$c_i = \sum_{j=1}^{s} a_{ij} \tag{3.42}$$

as discussed in section 3.3. Since $\hat{a}_{ij}$ are the coefficients of an explicit method, $\hat{A}$ must be strictly lower-triangular, and thus $\hat{a}_{1j} = 0 \forall j$. This means that $c_1 = 0$. Since a_{ij} are the coefficients of a DIRK method, $a_{1j} = 0 \forall j > 1$, so $a_{11} = c_1 = 0$. This means the DIRK method must be an EDIRK method. The most basic example of such an ERK-EDIRK pair is the first-order forward-backward Euler:

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1 & 0 \end{array} \quad \& \quad \begin{array}{c|cc} 0 & & \\ 1 & 0 & 1 \\ \hline & 0 & 1 \end{array}. \tag{3.43}$$

A second-order ERK-EDIRK pair can be constructed with the trapezoidal rule, adding no additional stages:

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \quad \& \quad \begin{array}{c|cc} 0 & & \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \tag{3.44}$$

ERK-ESDIRK pairs of orders three, four, and five are developed in [44]. The schemes are named ARK3(2)4L[2]SA, ARK4(3)6L[2]SA, and ARK5(4)8L[2]SA, in accordance with the naming convention ARK$q(p)sS[q_{SO}]X$ for a q^{th}-order method, paired with a p^{th}-order method, with s stages. S denotes a characterization of the stability, and q_{SO} is the stage-order of the implicit method. The letter X is used for any other trait of note. We will denote the methods from 3.43, 3.44, and [44] as ESDIRKn, where n is the order of the method. Thus ARK5(4)8L[2]SA will be called ESDIRK5, and so on.

These pairs are employed to form an IMEX method by calculating two seperate sets of stage values using the following equations:

$$M\hat{k}_i = f(u_n + \Delta t\hat{w}_i + \Delta t w_i) \tag{3.45}$$

$$Mk_i = g(u_n + \Delta t\hat{w}_i + \Delta t w_i) \tag{3.46}$$

where, w_i is defined in 3.13 and

$$\hat{w}_i = \sum_{j=1}^{s} \hat{a}_{ij}\hat{k}_j. \tag{3.47}$$

In the IMEX method, u_n is advanced to u_{n+1} by

$$u_{n+1} = u_n + \Delta t \sum_{i=1}^{s} \hat{b}_i\hat{k}_i + \Delta t \sum_{i=1}^{s} b_i k_i. \tag{3.48}$$

Since $\hat{A}$ comes from an ERK method, $\hat{a}_{ij} = 0$ unless $j < i$, so the sum in equation 3.47 might as well only go up to $i-1$. This means that equation 3.46 can be solved for k_i without knowing $\hat{k}_i$. As equation 3.45 cannot be solved for $\hat{k}_i$ without knowing k_i, there exists a natural order for calculating the $2s$ stage values, solving equation 3.46 for k_i, then equation 3.45 for $\hat{k}_i$, at each stage from $i = 1$ to $i = s$. We present this procedure as algorithm 1

3.6 Implicit-Explicit Prediction Methods

In practice, we use the RK method pairs ESDIRK3 and ESDIRK5 to build a predictor-corrector framework, as shown in [30], to advance the solution u_n forward in time. We extend this to predict the values of $\mathbf{W}$ of the IRK defined in equation 3.20 using the values w_i from a predictor RK method. In order to make effective predictions at each stage, it is required that $\hat{c} = c$, so that explicit prediction is made at a node of the implicit method, and not elsewhere in time. In this section we develop novel ERK-EDIRK pairs that can be used to predict values at each node of the Radau IIA and Lobatto IIIC methods introduced in 3.2. The Radau IIA method of s stages has nodes which are the zeros of the polynomial

$$\frac{d^{s-1}}{dx^{s-1}}\left(x^{s-1}(x-1)^s\right) \tag{3.49}$$

Algorithm 1: Implicit-Explicit (IMEX) Runge-Kutta Method (ERK-EDIRK Pair)

```
**Result:** ODE Solution $u_{n+1}$ at time $t_0 + \Delta t$
**Input:** IRK scheme $\hat{A}$, $\hat{b}$, $\hat{c}$
**Input:** EDIRK scheme $A$, $b$, $c = \hat{c}$
**Input:** ODE $Mu_t = f(t,u) + g(t,u)$
**Input:** Solution $u_n$ at time $t_n$
**Input:** Timestep $\Delta t > 0$
Set $\tilde{T} = T\left(\boldsymbol{U}_0^f + \Delta t \tilde{\boldsymbol{W}}^f\right)$;
**for** *i from 1 to s* **do**
    Calculate $\hat{w}_i = \sum_{j=1}^s \hat{a}_{ij}\hat{k}_j$;
    Define $w_i = \sum_{j=1}^s \hat{a}_{ij} k_j$;
    Solve $Mk_i = g(t_n + c_i\Delta t, u_n + \Delta t\hat{w}_i + \Delta t w_i)$ implicitly for $k_i$;
    Solve $M\hat{k}_i = f(t_n + c_i\Delta t, u_n + \Delta t\hat{w}_i + \Delta t w_i)$ explicitly for $\hat{k}_i$;
**end**
Set $u_{n+1} = u_n + \Delta t \sum_{i=1}^s \hat{b}_i\hat{k}_i + \Delta t \sum_{i=1}^s b_i k_i$;
```

while Lobatto methods have nodes at one, zero, and the zeros of $P'_{s-1}(x)$, where $P_n(x)$ is the n^{th} Legende Polynomial. To ensure consistency of the ERK-EDIRK pairs, we set the first node of our method to zero, and add an extra zero node at the beginning in the case of Radau IIA methods, where c_1 is non-zero. These nodes exactly prescribe those of the methods we develop here. The lowest-order ERK-EDIRK pair is found in equation 3.43. This pair works with both the first-order Radau IIA (Backward Euler), and the second-order Lobatto IIIC, shown in equations 3.26 and 3.27.

In order to have the best prediction at each time $t_n + c_i\Delta t$, we prioritize stage order at each stage for the EDIRK method. This contrasts with the usual priorities in choosing the coefficients of a RK method, where methods generally focus on high order and stability at $t + \Delta t$, at the expense of stage accuracy. To achieve accuracy at each stage, we first consider a generic RK method with lower-triangular coefficients A:

$$\begin{array}{c|cccc} c_1 & a_{11} & & & \\ c_2 & a_{21} & a_{22} & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \tag{3.50}$$

where each a_{ii} is possibly zero. This is either a DIRK method or an ERK method. Since our prescribed nodes require that $c_1 = 0$, q_1, as defined in equation 3.25, is arbitrarily high, since $u(t_n) = u_n$ and $w_1 = 0$. To determine the requirements for the subsequent stages, we define the stage $i > 1$ method of 3.50 as

$$\begin{array}{c|cccc} c_1 & a_{11} & & & \\ c_2 & a_{21} & a_{22} & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_i & a_{i1} & a_{i2} & \cdots & a_{ii} \\ \hline & a_{i1} & a_{i2} & \cdots & a_{ii} \end{array}. \tag{3.51}$$

This stage i method can be viewed as an RK method stepping u_n not from t_n to $t_n + \Delta t$, but from t_n to $t_n + \Delta t c_i$, yielding $u_n + \Delta t w_i$ as the result, as set out in 3.12. Comparing this result to 3.25, the order of the stage i method is precisely q_i, the order of each stage. To determine this value, we modify the order condition from equation 3.35 in section 3.3,

$$\sum b_i \Phi_i(\tau) = \frac{1}{\gamma(\tau)} \tag{3.52}$$

to adjust for the time goal of $t_n + \Delta t c_i$. This gives order conditions of

$$\sum a_{ij} \Phi_j(\tau) = \frac{c_i^q}{\gamma(\tau)} \tag{3.53}$$

where q is the order of the tree τ.

Firstly, this implies $a_{11} = c_1 = 0$. We now use these modified order conditions to optimize the order of the stage methods of the EDIRK method sequentially, beginning by applying equation 3.34 of the stage $i = 2$ method. This gives

$$c_2 = \sum_{j=1}^{2} a_{2j} = a_{21} + a_{22}, \tag{3.54}$$

while the second order condition from 3.53 gives

$$\frac{c_2^2}{2} = \sum_{j=1}^{2} a_{2j} c_j = a_{21} c_1 + a_{22} c_2 = a_{22} c_2, \tag{3.55}$$

implying that $a_{22} = \frac{1}{2} c_2$. Together, these two equations require that $a_{21} = a_{22} = \frac{1}{2} c_2$, embedding the implicit trapezoid method in the first two stages of the DIRK method. The first two rows of and A are thus uniquely fixed:

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & \frac{1}{2}c_2 & \frac{1}{2}c_2 & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_i & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_{s1} & b_{s2} & \cdots & b_{ss} \end{array} \tag{3.56}$$

The order conditions of the stage $i = 3$ method are

$$c_3 = \sum_{j=1}^{3} a_{3j} \qquad \frac{c_3^2}{2} = \sum_{j=1}^{3} a_{3j}c_j \qquad \frac{c_3^3}{3} = \sum_{j=1}^{3} a_{3j}c_j^2 \qquad \frac{c_3^3}{6} = \sum_{j=1}^{3}\sum_{k=1}^{3} a_{3j}a_{jk}c_k. \tag{3.57}$$

It appears there are four equations for three variables. However, as $c_1 = 0$, the middle equations simplify to

$$\frac{c_3^2}{2} = a_{32}c_2 + a_{33}c_3 \qquad \frac{c_3^3}{3} = a_{32}c_2^2 + a_{33}c_3^2. \tag{3.58}$$

Multiplying the left equation by a_{33} and the right equation by $\frac{1}{2}$,

$$\frac{a_{33}c_3^2}{2} = a_{32}a_{33}c_2 + a_{33}^2c_3 \qquad \frac{c_3^3}{6} = \frac{a_{32}c_2^2}{2} + \frac{a_{33}c_3^2}{2}. \tag{3.59}$$

Substituting the right side of the left equation for the last term of the right equation, we have

$$\frac{c_3^3}{6} = \frac{a_{32}c_2^2}{2} + a_{32}a_{33}c_2 + a_{33}^2c_3. \tag{3.60}$$

If the first two rows of the Butcher tableau are consistent with equation 3.56, then this is simply the fourth equation in 3.57. Therefore the EDIRK stage 3 method is able to achieve third-order accuracy with uniquely determined coefficients. The first three rows of coefficients for the ESDIRK method are

$$\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & \frac{1}{2}c_2 & \frac{1}{2}c_2 & & & \\
c_3 & c_3\left(1 - \frac{c_3^2+2c_2c_3^2-3c_2}{6c_2(1-c_3)}\right) & \frac{-c_3^3}{6c_2(1-c_3)} & \frac{3c_3-2c_3^2}{6(1-c_3)} & & \\
\vdots & \vdots & \vdots & \vdots & \ddots & \\
c_i & a_{s1} & a_{s2} & a_{s3} & \cdots & a_{ss} \\
\hline
 & b_{s1} & b_{s2} & b_{s3}\cdots & b_{ss} &
\end{array} \tag{3.61}$$

The weights, b_i, are set equal to the last row of the coefficients, as this maximizes the order of accuracy of the method. Applying this procedure to the third-order Radau IIA method

$$\begin{array}{c|cc}
\frac{1}{3} & \frac{5}{12} & \frac{-1}{12} \\
1 & \frac{3}{4} & \frac{1}{4} \\
\hline
 & \frac{3}{4} & \frac{1}{4}
\end{array} \tag{3.62}$$

we get the ESDIRK method

$$\begin{array}{c|ccc}
0 & 0 & 0 & 0 \\
\frac{1}{3} & \frac{1}{6} & \frac{1}{6} & 0 \\
1 & 0 & \frac{3}{4} & \frac{1}{4} \\
\hline
 & 0 & \frac{3}{4} & \frac{1}{4}
\end{array} \tag{3.63}$$

with stage order two, and third-order accuracy at stage three. From the nodes of the fourth-order Lobatto IIIC method

$$\begin{array}{c|ccc} 0 & \frac{1}{6} & -\frac{1}{3} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{6} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \tag{3.64}$$

the procedure gives

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \tag{3.65}$$

which has stage order two, and third-order accuracy at stage three. We note that the order conditions from the branchless trees in equation 3.52 require that the final row of coefficients and weights match those of the method from which the ESDIRK is produced.

In general, it is not possible to proceed this way indefinitely, achieving i^{th}-order accuracy at the stage i method. For fourth-order accuracy, equation 3.53 imposes four additional conditions beyond those for third-order accuracy, and seven more for fifth-order accuracy. These are too many for the number of degrees of freedom, i, in the row of coefficients. Extending the procedure to the fifth-order Radau IIA method

$$\begin{array}{c|ccc} \frac{4-\sqrt{6}}{10} & \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{4-\sqrt{6}}{10} & \frac{296-169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ 1 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \\ \hline & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array} \tag{3.66}$$

and using the weights from the method as the final row of the ESDIRK method coefficients, we get the method

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{4-\sqrt{6}}{10} & \frac{4-\sqrt{6}}{20} & \frac{4-\sqrt{6}}{20} & 0 & 0 \\ \frac{4+\sqrt{6}}{10} & \frac{-2-3\sqrt{6}}{75} & \frac{528+217\sqrt{6}}{1800} & \frac{48+7\sqrt{6}}{360} & 0 \\ 1 & 0 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \\ \hline & 0 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array} \tag{3.67}$$

which achieves stage order two, but third-order accuracy at stages three and four. For the sixth-order Lobatto IIIC method

$$\begin{array}{c|cccc}
0 & \frac{1}{12} & -\frac{\sqrt{5}}{12} & \frac{\sqrt{5}}{12} & -\frac{1}{12} \\
\frac{5-\sqrt{5}}{10} & \frac{1}{12} & \frac{1}{4} & \frac{10-7\sqrt{5}}{60} & \frac{\sqrt{5}}{60} \\
\frac{5+\sqrt{5}}{10} & \frac{1}{12} & \frac{10+7\sqrt{5}}{60} & \frac{1}{4} & -\frac{\sqrt{5}}{60} \\
1 & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12} \\
\hline
 & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12}
\end{array} \tag{3.68}$$

we deviate from the procedure described above, exchanging one order of accuracy at the third stage for an extra order at the fourth. The resulting ESDIRK,

$$\begin{array}{c|cccc}
 & & & & \\
\frac{5-\sqrt{5}}{10} & \frac{5-\sqrt{5}}{20} & \frac{5-\sqrt{5}}{20} & & \\
\frac{5+\sqrt{5}}{10} & \frac{-5+3\sqrt{5}}{60} & \frac{1+\sqrt{5}}{6} & \frac{25-7\sqrt{5}}{60} & \\
1 & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12} \\
\hline
 & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12}
\end{array} \tag{3.69}$$

has stage order two, but fourth-order accuracy at stage four.

To complete each pair, we create an ERK method with nodes and weights matching those of the corresponding ESDIRK and IRK methods. We seek to maximize the order of each ERK method, by satisfying as many of the order conditions from equation 3.52. In some cases, we find there remains a degree of freedom, even when the maximum number of order conditions possible has been satisfied. In this case, we adjust this degree of freedom according to our secondary consideration of minimizing the quantity

$$\max_{i,j} \left| \frac{\hat{a}_{ij} - a_{ij}}{a_{ii}} \right| \tag{3.70}$$

to reduce numerical error when using the IMEX method introduced in [78], shown in equation 5.17. Finally, we have the following ERK-ESDIRK-IRK triplets, where the ERK-ESDIRK pair serves as a prediction method for the IRK:

First-order Radau IIA, with first-order ERK:

$$\begin{array}{c|cc}
0 & & \\
1 & 1 & \\
\hline
 & 1 & 0
\end{array}
\quad \& \quad
\begin{array}{c|cc}
0 & & \\
1 & 0 & 1 \\
\hline
 & 0 & 1
\end{array}
\quad \& \quad
\begin{array}{c|c}
1 & 1 \\
\hline
 & 1
\end{array} \tag{3.71}$$

Second-order Lobatto IIIC, with first-order ERK:

$$\begin{array}{c|cc}
0 & & \\
1 & 1 & \\
\hline
 & 1 & 0
\end{array}
\quad \& \quad
\begin{array}{c|cc}
0 & & \\
1 & \frac{1}{2} & \frac{1}{2} \\
\hline
 & \frac{1}{2} & \frac{1}{2}
\end{array}
\quad \& \quad
\begin{array}{c|cc}
0 & \frac{1}{2} & -\frac{1}{2} \\
1 & \frac{1}{2} & \frac{1}{2} \\
\hline
 & \frac{1}{2} & \frac{1}{2}
\end{array} \tag{3.72}$$

Third-order Radau IIA, with third-order ERK:

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{3} & \frac{1}{3} & & \\ 1 & -1 & 2 & \\ \hline & 0 & \frac{3}{4} & \frac{1}{4} \end{array} \quad \& \quad \begin{array}{c|ccc} 0 & & & \\ \frac{1}{3} & \frac{1}{6} & \frac{1}{6} & \\ 1 & 0 & \frac{3}{4} & \frac{1}{4} \\ \hline & 0 & \frac{3}{4} & \frac{1}{4} \end{array} \quad \& \quad \begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & \frac{-1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array} \tag{3.73}$$

Fourth-order Lobatto IIIC, with third-order ERK:

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \quad \& \quad \begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \quad \& \quad \begin{array}{c|ccc} 0 & \frac{1}{6} & -\frac{1}{3} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{6} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \tag{3.74}$$

Fifth-order Radau IIA, with third-order ERK method (IRK omitted for brevity):

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{4-\sqrt{6}}{10} & \frac{4-\sqrt{6}}{10} & & & \\ \frac{4+\sqrt{6}}{10} & \frac{-28-17\sqrt{6}}{100} & \frac{68+27\sqrt{6}}{100} & & \\ 1 & \frac{-15+6\sqrt{6}}{8} & \frac{7-2\sqrt{6}}{8} & \frac{16-4\sqrt{6}}{8} & \\ \hline & 0 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array} \quad \& \quad \begin{array}{c|cccc} 0 & & & & \\ \frac{4-\sqrt{6}}{10} & \frac{4-\sqrt{6}}{20} & \frac{4-\sqrt{6}}{20} & & \\ \frac{4+\sqrt{6}}{10} & \frac{-2-3\sqrt{6}}{75} & \frac{528+217\sqrt{6}}{1800} & \frac{48+7\sqrt{6}}{360} & \\ 1 & 0 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \\ \hline & 0 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array} \tag{3.75}$$

Sixth-order Lobatto IIIC, with fourth-order ERK method (IRK omitted for brevity):

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{5-\sqrt{5}}{10} & \frac{5-\sqrt{5}}{10} & & & \\ \frac{5+\sqrt{5}}{10} & -\frac{\sqrt{5}}{10} & \frac{5+2\sqrt{5}}{10} & & \\ 1 & \frac{-13+7\sqrt{5}}{12} & \frac{-5+3\sqrt{5}}{12} & \frac{15-5\sqrt{5}}{6} & \\ \hline & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12} \end{array} \quad \& \quad \begin{array}{c|cccc} 0 & & & & \\ \frac{5-\sqrt{5}}{10} & \frac{5-\sqrt{5}}{20} & \frac{5-\sqrt{5}}{20} & & \\ \frac{5+\sqrt{5}}{10} & \frac{-5+3\sqrt{5}}{60} & \frac{1+\sqrt{5}}{6} & \frac{25-7\sqrt{5}}{60} & \\ 1 & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12} \\ \hline & \frac{1}{12} & \frac{5}{12} & \frac{5}{12} & \frac{1}{12} \end{array} \tag{3.76}$$

Chapter 4

Spatial Discretization and Solvers

4.1 Spatial Discretization

From the governing equations found in chapter 2, we build vector-valued ordinary differential equations in N variables, with the vector value expressing an approximation of the solution. For this we use finite element methods, specifically Galerkin finite element methods. These offer many advantages, including high order accuracy and applicability to flexible and irregular meshes.

4.1.1 Fluid Spatial Discretization

Our approach to discretizing the fluid equations described in the previous chapter is a high-order discontinuous Galerkin (DG) formulation. We split the fluid domain, Ω, into triangular mesh elements and impose a set of nodal basis functions of third-order polynomials. At the boundary of these elements, we compute inviscid fluxes with Roe's Method as introduced in [63], and use the compact DG method in [58] to compute numerical fluxes for the viscous terms.

We use the standard procedure for DG discretization of governing equations with second derivatives found in [3], namely the conversion to a system of first-order equations via an auxiliary gradient variable $\boldsymbol{q} = \nabla \boldsymbol{u}$. Then the fluid is governed by

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}^i(\boldsymbol{u}) - \nabla \cdot \boldsymbol{F}^v(\boldsymbol{u}, \boldsymbol{q}) = 0, \quad \boldsymbol{q} = \nabla \boldsymbol{u} \tag{4.1}$$

where $\boldsymbol{F}^i$ is the inviscid portion of $\boldsymbol{F}$ and $\boldsymbol{F}^v$ is the viscous component.

As is standard with any finite element method, we seek a solution within the space of polynomial functions of degree up to p on each element K in the set of elements $\mathcal{T}_h$ specified by the mesh we impose on the fluid domain. This solution space $V_h^p \times \Sigma_h^p$ is defined by

$$V_h^p = \left\{ \boldsymbol{\nu} \in [L^2(\Omega)]^4 | \ \boldsymbol{\nu}|_k \in [\mathcal{P}_p(K)]^4 \ \forall K \in \mathcal{T}_h \right\} \tag{4.2}$$

$$\Sigma_h^p = \left\{ \boldsymbol{\tau} \in [L^2(\Omega)]^{4\times 2} | \; \boldsymbol{\tau}|_k \in [\mathcal{P}_p(K)]^{4\times 2} \; \forall K \in \mathcal{T}_h \right\}. \tag{4.3}$$

Multiplying equation 4.1 by arbitrary test functions $\boldsymbol{\nu} \in V_h^p$ and $\boldsymbol{\tau} \in \Sigma_h^p$ and integrating by parts, we obtain our semi-discrete DG formulation, with solution $\boldsymbol{u}_h \in V_h^p$ and $\boldsymbol{q}_h \in \Sigma_h^p$ such that

$$\mathbf{0} = \int_K \frac{\partial \boldsymbol{u}_h}{\partial t} \cdot \boldsymbol{\nu} \; dx + \int_K \left(\boldsymbol{F}^i(\boldsymbol{u}_h) - \boldsymbol{F}^\nu(\boldsymbol{u}_h, \boldsymbol{q}_h) \right) : \nabla \boldsymbol{\nu} \; dx - \oint_{\partial K} \left(\widehat{\boldsymbol{F}^i(\boldsymbol{u}_h)} - \widehat{\boldsymbol{F}^v(\boldsymbol{u}_h, \boldsymbol{q}_h)} \right) \cdot \boldsymbol{\nu} \; ds \tag{4.4}$$

$$\mathbf{0} = \int_K \boldsymbol{q}_h : \boldsymbol{\tau} \; dx + \int_K \boldsymbol{u}_h \cdot (\nabla \cdot \boldsymbol{\tau}) \; dx - \oint_{\partial K} (\hat{\boldsymbol{u}}_h \otimes \boldsymbol{n}) : \boldsymbol{\tau} \; ds \tag{4.5}$$

for all $K \in \mathcal{T}_h$, $\boldsymbol{\nu} \in [\mathcal{P}_p(K)]^4$, and $\boldsymbol{\tau} \in [\mathcal{P}_p(K)]^{4\times 2}$, using numerical fluxes $\widehat{\boldsymbol{F}^i(\boldsymbol{u}_h)}$ on the boundary ∂K computed according to Roe's method [63], modified for the ALE formulation described in [60] for inviscid terms, and, for viscuous terms, numerical fluxes $\widehat{\boldsymbol{F}^v(\boldsymbol{u}_h, \boldsymbol{q}_h)}$ and $\hat{\boldsymbol{u}}_h$ computed according to the Compact Discontinuous Galerkin method described in [58]. The normal vector $\boldsymbol{n}$ is defined at the boundary ∂K.

To compute these numerical functions, we for each boundary e between two neighboring elements K and K', we denote the solution in element K is $\boldsymbol{u}_h$ and the solution on the other side of e, in element K', is $\boldsymbol{u}_h'$. We define a *switch function* $S_K^{K'} \in \{-1, 1\}$ with the requirement that $S_K^{K'} = -S_{K'}^K$. To satisfy this, we order all elements and set $S_K^{K'} = -1$ when K comes after K', and $S_K^{K'} = 1$ when K comes before K'. When $S_{K'}^K = 1$, we set $\hat{\boldsymbol{u}}_h = \boldsymbol{u}_h'$. When $S_{K'}^K = -1$, we set $\hat{\boldsymbol{u}}_h = \boldsymbol{u}_h$, as is standard up-winding practice.

With this numerical flux $\hat{\boldsymbol{u}}_h$, we define the piecewise function on K

$$\hat{\boldsymbol{u}}_h^e = \begin{cases} \hat{\boldsymbol{u}}_h & \text{on face } e \\ \boldsymbol{u}_h & \text{on interior of } K \end{cases} \tag{4.6}$$

and similarly on element K',

$$\hat{\boldsymbol{u}}_h^{e\,\prime} = \begin{cases} \hat{\boldsymbol{u}}_h & \text{on face } e \\ \boldsymbol{u}_h & \text{on interior of } K' \end{cases}. \tag{4.7}$$

Using this equation, we define the face gradient $\boldsymbol{q}_h^e$ of element K by with a slight variation on equation 4.5:

$$\mathbf{0} = \int_K \boldsymbol{q}_h^e : \boldsymbol{\tau} \; dx + \int_K \boldsymbol{u}_h \cdot (\nabla \cdot \boldsymbol{\tau}) \; dx - \oint_{\partial K} (\hat{\boldsymbol{u}}_h^e \otimes \boldsymbol{n}) : \boldsymbol{\tau} \; ds \tag{4.8}$$

and $\boldsymbol{q}_h^{e\,\prime}$ on element K' by

$$\mathbf{0} = \int_{K'} \boldsymbol{q}_h^{e\,\prime} : \boldsymbol{\tau} \; dx + \int_K \boldsymbol{u}_h \cdot (\nabla \cdot \boldsymbol{\tau}) \; dx - \oint_{\partial K'} \left(\hat{\boldsymbol{u}}_h^{e\,\prime} \otimes \boldsymbol{n} \right) : \boldsymbol{\tau} \; ds. \tag{4.9}$$

Finally, we define the viscous numerical fluxes $\widehat{\boldsymbol{F}^v(\boldsymbol{u}_h, \boldsymbol{q}_h)}$ are defined on the face e by

$$\widehat{\boldsymbol{F}^v(\boldsymbol{u}_h, \boldsymbol{q}_h)} = C_{11}\left(\boldsymbol{u}_h' - \boldsymbol{u}_h\right) + \begin{cases} \boldsymbol{F}^v\left(\boldsymbol{u}_h^e, \boldsymbol{q}_h\right) \cdot \boldsymbol{n} & \text{when } S_{K'}^{K} = 1 \\ \boldsymbol{F}^v\left(\boldsymbol{u}_h^{e\,\prime}, \boldsymbol{q}_h{}'\right) \cdot \boldsymbol{n} & \text{when } S_{K}^{K'} = 1 \end{cases} \tag{4.10}$$

with the stabilization parameter C_{11} set to the value of $\frac{10}{h_{\min}}$ where $h_{\min}$ is the height of the element with respect to the face e. Having defined the numerical fluxes at interfaces between all elements, we choose numerical fluxes at the boundary of Ω to weakly enforce boundary conditions, according to [57].

With these definitions in place, we impose a nodal basis on equations 4.5 and 4.4. First, we define a set of nodes x_j on each element K. Using Lagrange interpolation functions $\phi(x) \in \mathcal{P}_p(K)$, with the property $\phi_i(x_j) = \delta_{ij}$ as our basis functions, the solution in each element K can be written as a linear combination of these interpolation functions, with coefficients u_i:

$$\boldsymbol{u}_h(x) = \sum_{i=1}^{n} u_i \phi_i(x) \tag{4.11}$$

Similarly the auxiliary variable $\boldsymbol{q}_h$, the time derivative $\frac{\partial \boldsymbol{u}_h}{\partial t}$, and the test functions ν and τ can be described as a linear combination of these interpolation functions. Applying this equations 4.5 and 4.4 and integrating with high-order Gaussian quadrature, we get one equation to satisfy for each test function. As the Lagrange interpolation functions form a (nodal) basis for the space of test functions, the set of equations derived from all possible test functions is equivalent to the set of equations derived from setting the test functions to each interpolation function. From there, we can eliminate the coefficients for the auxiliary variable $\boldsymbol{q}_h$ from the system of equations, and obtain a semi-discrete Ordinary Differential Equation in the vector space of sets of coefficients u_i, with solution vectors $\boldsymbol{u}^f$:

$$\boldsymbol{M}^f \frac{d\boldsymbol{u}^f}{dt} = \boldsymbol{r}^f\left(\boldsymbol{u}^f\right) \tag{4.12}$$

with mass matrix $\boldsymbol{M}^f$ and residual function $\boldsymbol{r}^f\left(\boldsymbol{u}^f\right)$.

As each of the components of the solution vector $\boldsymbol{u}^f$ corresponds to one of the interpolation functions, and each of the interpolation functions corresponds to one of the nodes in one of the elements, $\boldsymbol{u}^f$ has N components, where N is the product of the number of nodes per element, the number components in the governing equation, and the number of elements. The number of components is three in the one-dimensional case, four in the two-dimensional case, and five in the case of three dimensions. The number of nodes per element is given in table 4.1.1, and depends on the order of the polynomial space used. The number of elements can vary widely, but we use up to several thousand in some cases. These factors combine to give a values of N up to hundreds of thousands for two dimensions, and easily into the millions for three.

Another consequence of using the nodal basis of interpolation functions as test functions is that each component of the output of residual function $\boldsymbol{r}^f$ corresponds to a single node in one element. The interior integrals appearing in equations 4.4 and 4.5 cause this output component to depend on all components of the input corresponding to the nodes in the same element. The boundary integrals appearing in equations 4.4 and 4.5, together with the numerical flux functions chosen, mean that the output components depend on input components from neighboring elements, but not on non-neighboring elements.

Because each of the test functions τ (and the eliminated ν) have support limited to one element, the product of any two test functions is zero throughout the domain unless they come from the same element. This implies that the mass matrix $\boldsymbol{M}^f$ is block diagonal, with one block for each element. This fact, together with the dependence of $\boldsymbol{r}^f$ output components being limited to an element and its neighbors, are helpful properties for the parallel implementation described in section 4.2

Polynomial Order	Triangular Elements	Quadrilateral Elements
p=2	6	9
p=3	10	16
p=4	15	25
p=5	21	36
p=6	28	49

Table 4.1: Number of higher order nodes per element for triangles and quadrangles in two dimensions

4.1.2 Structure Spatial Discretization

In contrast with the fluid, we use a continuous Galerkin finite element method to model flexible structures. Breaking the domain into triangles, we fit curved boundaries with isoparametric elements. The solution space is the set of continuous piecewise polynomials of degree p on the mesh $\mathcal{T}_h$:

$$V_h^p = \left\{ \boldsymbol{\nu} \in [\mathcal{C}_0(\Omega)] \mid \nu|_K \in [\mathcal{P}_p(K)]^2 \ \forall K \in \mathcal{T}_h \right\} \tag{4.13}$$

We then define two subspaces. The first,

$$V_{h,D}^p = \{ \nu \in V_h^p | \ \nu|_{\Gamma_D} = x_D \}, \tag{4.14}$$

adheres to the non-homogeneous Dirichlet boundary conditions, and the second adheres to homogeneous Dirichlet boundary conditions:

$$V_{h,0}^p = \{ \nu \in V_h^p | \ \nu|_{\Gamma_D} = 0 \} \tag{4.15}$$

We proceed by multiplying the equation described in the previous chapter by any test function $\boldsymbol{z}$ from $V_{h,0}^p$ and integrating over the domain V. In applying Green's theorem, we arrive at the finite element formulation, a solution $\boldsymbol{x}_h \in V_{h,D}^p$ such that for all test functions $\boldsymbol{z} \in V_{h,0}^p$,

$$\int_V \rho \frac{\partial^2 \boldsymbol{x}_h}{\partial t^2} \boldsymbol{z} dX = -\int_V \boldsymbol{P}\left(\boldsymbol{F}(\boldsymbol{x}_h)\right) : \nabla \boldsymbol{z} dX + \oint_{\Gamma_N} \boldsymbol{t}(\boldsymbol{x}_h) \cdot \boldsymbol{z} dS + \int_V \boldsymbol{b} \cdot \boldsymbol{z} dX. \tag{4.16}$$

As with the fluid, these integrals are calculated with high-order Gaussian quadrature, and a nodal basis is used to produce a discrete solution vector $\boldsymbol{Y}$ of coefficients y_i multiplied by basis function ϕ_i to form the solution. With this, we form the global discrete residual vector $R(Y)$ and the second-order ODE

$$\boldsymbol{M} \frac{d^2 \boldsymbol{Y}}{dt^2} = \boldsymbol{R}(\boldsymbol{Y}) \tag{4.17}$$

which we transform into a first-order system with a velocity vector $\boldsymbol{V} = \frac{d\boldsymbol{Y}}{dt}$. The combined solution vector $\boldsymbol{u}^s$ is the concatenation of these two solution vectors:

$$\boldsymbol{u}^s = \begin{bmatrix} \boldsymbol{Y} \\ \boldsymbol{V} \end{bmatrix} \tag{4.18}$$

and the structure residual is

$$\boldsymbol{r}^s\left(\boldsymbol{u}^s\right) = \begin{bmatrix} \boldsymbol{V} \\ \boldsymbol{R}(\boldsymbol{Y}) \end{bmatrix} \tag{4.19}$$

and the mass matrix is of block form

$$\boldsymbol{M}^s = \begin{bmatrix} \boldsymbol{I} & 0 \\ 0 & \boldsymbol{M} \end{bmatrix}. \tag{4.20}$$

These combine to give the semi-discrete form of the continuous Galerkin formulation for the structure:

$$\boldsymbol{M}^s \frac{d\boldsymbol{u}^s}{dt} = \boldsymbol{r}^s(\boldsymbol{u}^s). \tag{4.21}$$

4.2 Fluid Solvers

As Discussed in 3.2, our use of Runge-Kutta methods requires three types of solvers. For ERK methods, we solve for the stage values k_i^f explicitly with a relatively straightforward linear solve. Though we solve for k_1^f in the same manner when employing ESDIRK methods, later stages, and all stages in DIRK methods, require non-linear solves of size N, where N is the number of degrees of freedom defined in 4.1.1. Finally, IRK methods require non-linear solves of size $s \cdot N$, where s is the number of stages. These non-linear solves can be very large and computationally expensive.

To efficiently solve these non-linear equations, we take a parallel approach in all but the simplest cases. First, the fluid portion of the computational domain is decomposed into

n_{proc} partitions, the number of processors to be used. This is done by building the adjacency graph, representing elements as vertices and connecting two vertices if the corresponding elements share an edge. The partitions are then created using the METIS software described in [43]. The n_{proc} partitions created generally have roughly the same number of nodes, while minimizing the number of edges shared between elements in two different partitions. This allows the partitions to be as self-contained as possible, reducing the need for communication between processors.

4.2.1 Explicit Parallel Fluid Solver

The stage value k_i^f of an explicit stage i depends only on the current state u_n^f and the previous stage values $k_j^f, j < i$, by the equation

$$M^f k_i^f = r^f \left(u_n^f + \Delta t w_i^f\right), \ w_i^f = \sum_{j=1}^{i-1} a_{ij} k_j^f. \tag{4.22}$$

Note that the boldface has been dropped from the notation, as we are dealing with only one stage. Additionally, time dependence is taken out of the notation for simplicity.

As noted in section 4.1.1, the mass matrix introduced in equation 4.12 and found again in equation 4.22 is block diagonal, with each block corresponding to an element of the fluid domain. This allows for a natural parallelism, with each block factored and stored in the appropriate processor, according to the partition created. The inverse of M^f can then be applied without communication between processors. The residual function r^f does not have this property, as the flux functions from section 4.1.1 require values from neighboring elements. However, each process only needs the up-to-date values from first-degree neighbors, and no others beyond this single layer beyond the partition the process is assigned.

To solve equation 4.22, we first evaluate r^f at all elements which have neighbors outside their partitions. We then initiate asynchronous send and receive operations with the results, communicating between processors. Then, r^f is calculated on the remaining elements, which requires no communication between processors. Once the communication and local residual evaluation is finished, the inverse of M^f is applied at each element, resulting in k_i^f.

4.2.2 Single-Stage Parallel Newton-Krylov Fluid Solvers

When using DIRK, EDIRK, and ESDIRK methods, a stage value k_i^f of an implicit stage i depends on the current state u_n^f, previous stage values $k_j^f, j < i$, and k_i^f itself, by the equation

$$M^f k_i^f = r^f \left(u_n^f + \Delta t w_i^f\right), \ w_i^f = \sum_{j=1}^{i} a_{ij} k_j^f. \tag{4.23}$$

This requires a non-linear solve, for which we use Newton's Method. This requires linear solves of the form

$$\left(M^f - a_{ii}\Delta t\frac{dr^f}{du^f}\right)\delta k_i^f = r^f \tag{4.24}$$

for which we use an iterative preconditioned GMRES method. The matrix on the left can become quite large, and we would prefer to calculate its values during the computation of matrix-vector products, on an as-needed basis, using the so-called matrix-free method. Unfortunately, this creates significant difficulties for preconditioning.

We compute the matrix and store it in a block-diagonal format, according to the 3DG implementation found in [59]. In addition to the blocks on the diagonal found in the sparsity pattern of M^f, this consists of two off-diagonal blocks for each interface between two elements. Each processor is assigned the rows corresponding to elements in its partition, but not elements of neighboring partitions. The required matrix-vector products are thus computed with some communication between processors to transfer data associated with neighboring elements across the partition boundaries.

To precondition equation 4.24, we have a choice of preconditioners. The *block Jacobi* preconditioner has the same sparsity pattern as M^f and consists of the inverse of each diagonal block of the matrix. As this does not consider off-diagonal blocks, it requires no communication between processors to calculate in parallel implementation. The more expensive *block ILU(0)* preconditioner, described in [56], is an approximation of the LU factorization of the matrix using a modified Gaussian elimination strategy on the blocks of the matrix. Instead of allowing for fill, the block sparsity pattern of $L + U$ is kept at that of the original matrix. This limits the size of the storage required for $L + U$ to that of the original matrix.

Applied to the entire matrix, the block ILU(0) preconditioner, unlike the Block Jacobi, requires communication between processors, as the off-diagonal blocks are considered. We apply the block ILU(0) preconditioner individually to the rows held within each processor. This does not take off-diagonal blocks corresponding to dependence between partitions into account, eliminating the need for communication between processors, but increasing the number of GMRES iterations necessary when many processors are used. In fact, the ILU(0) we use is precisely block Jacobi when the number of processors is equal to the number of elements.

As in [56], we impose the requirement our meshes that if two elements K_1' and K_2' are neighbors of element K, then K_1' and K_2' are not neighbors. With this satisfied, the process for calculating the block ILU(0) is shown in algorithm 2. The algorithm for block matrices not satisfying this condition can be found in [56].

Algorithm 2: Block ILU(0) preconditioner

```
Result: Block LU factorization of preconditioner, $\tilde{L}\tilde{U}$
Input: Block sparse matrix $B$ with $T$ diagonal blocks
for $i$ from 1 to $T$ do
    for neighbors $j$ of $i$ with $j > i$ do
        $B_{ji} \leftarrow B_{ji}B_{ii}^{-1}$;
        $B_{jj} \leftarrow B_{jj} - B_{ji}B_{ij}$;
    end
end
$\tilde{L} \leftarrow I +$ strict block lower triangular portion of $B$;
$\tilde{U} \leftarrow$ block upper triangular portion of $B$
```

4.2.3 Multi-Stage Parallel Newton-Krylov Fluid Solvers

Though ERK and DIRK methods allow for the sequential calculation of stages, IRK methods do not. All stages must be calculated simultaneously, resulting in a non-linear system in $s \cdot N$ variables that we solve using Newton's method, as with individual stages. The resulting linear system, found in equation 3.24, is

$$\left(A^{-1} \otimes M^f - \boldsymbol{J}\left(u_n^f + \Delta t \boldsymbol{W}^f\right)\right) \delta \boldsymbol{W}^f = \boldsymbol{r}^f\left(u_n^f + \Delta t \boldsymbol{W}^f\right). \tag{4.25}$$

We again use a preconditioned GMRES method. The sparsity pattern of the matrix in equation 4.25 differs from the one found in equation 4.24 due the $s^2 - s$ off-diagonal mass matrices. Though the use of a block ILU(0) preconditioner which takes these mass blocks into account is a possible choice, adopt the *stage-uncoupled, shifted ILU(0)* preconditioner introduced in [56]. This takes the form

$$\begin{bmatrix} \tilde{L}_1\tilde{U}_1 & 0 & \cdots & 0 \\ 0 & \tilde{L}_2\tilde{U}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \tilde{L}_s\tilde{U}_s \end{bmatrix} \tag{4.26}$$

where $\tilde{L}_i\tilde{U}_i$ is the block ILU(0) factorization, calculated by algorithm 2, of the shifted matrix

$$\left(\left(A^{-1}\right)_{ii} + \alpha_i\right) M^f - \Delta t J_i \tag{4.27}$$

where J_i is the Jacobian of the i^{th} stage, and the i^{th} diagonal stage-block of $\boldsymbol{J}$. For the shift, α_i, we use the value adopted in [56],

$$\alpha_i = \sum_{j \neq i} \left|\left(A^{-1}\right)_{ji}\right| \tag{4.28}$$

with similar success.

When solving the full IRK system, here are certain advantages to the stage-uncoupled, shifted ILU(0) over the stage-coupled ILU(0) used in 4.2.2. Firstly, its implementation is trivial when algorithm 2 has already been implemented, as it involves only s iterations of this algorithm. Secondly, the memory needed to store the preconditioner scales as s times the memory needed for one block ILU(0) factorization, with no s^2 term. Finally, this choice offers the ability to use an alternate partitioning, with a factor s fewer partitions in the fluid domain, reducing the number of GMRES iterations necessary. This is done by handling each stage for each partition in a different processor, so that n_{proc} is equal to the number of partitions times s, instead of simply the number of partitions. This is known as *stage parallelization*, and results in no extra communication between processors, as no communication is necessary between stages for the stage-uncoupled, shifted ILU(0) preconditioner.

In practice, we do not take advantage of stage parallelization for fluid-structure interaction simulations. As we discuss further in section 5.3, we develop an algorithm that uses both EDIRK and IRK methods for each timestep. Though stage parallelization is possible, and beneficial, in the implementation of a pure IRK fluid solver, it is impossible with an EDIRK method, where stages are computed sequentially. Thus, switching between two mesh partitions would be necessary, one for the EDIRK, and another for the IRK, incurring very large communication costs.

4.2.4 Fluid Solver Computational Costs

As we use preconditioned Krylov solvers for the linear systems arising from Newton's method both with individually computed diagonally implicit stages, and with concurrently computed IRK stages, the two main costs of these solvers are computing the residual, stage Jacobians, and preconditioner, and applying the matrix at each GMRES iteration to calculate the matrix-vector product. For single-stage solves, one stage Jacobian, one ILU(0) preconditioner, and one residual are computed at each Newton iteration. The number of matrix applications is equal to the number of GMRES iterations. For multi-stage solves with s stages, each Newton iteration requires the computation of s stage Jacobians and residuals, as well as a preconditioner. Since we employ a stage-uncoupled preconditioner, each stage preconditioner $\tilde{L}_i\tilde{U}_i$ is exactly equivalent in cost to the preconditioner in the single-stage case. As there are s preconditioners needed, this means the calculation cost incurred at each Newton iteration of the multi-stage fluid solver is exactly s times that of a single-stage solver, not counting the matrix applications.

As shown in [56], the transformation in equation 3.22 reduces the cost of the matrix-vector products calculated in GMRES iterations from s^2 times that of the single-stage solver to nearly s times. The application of the multi-stage preconditioner is exactly s times that of the single-stage preconditioner, due to its block-diagonal structure. The multi-stage matrix itself, however, is not block diagonal, but has off-diagonal blocks of scaled mass matrices. These additional blocks require application of the mass matrix and scaling at each matrix application. As noted in 4.2.1, the mass matrix is block diagonal, with each block corresponding to an element in the mesh. Because we do not take advantage of stage parallelization, each partition contains the data for every element at each stage, as well as the portion of the mass matrix corresponding to those elements. This means that the multiplication by off-diagonal mass blocks in multi-stage matrix applications is done locally, incurring very little computational cost.

In comparing the cost of our single-stage to our multi-stage fluid solvers, we neglect the effect of the off-diagonal mass blocks, and count both the per-Newton-iteration assembly cost and per-GMRES-iteration matrix application cost as differing by a factor of s. In practice, the difference between these two costs can vary wildly, but we have found that in the applications studied here the Jacobian and preconditioner assembly cost incurred at each Newton iteration is typically between ten and a hundred times the cost of a GMRES iteration, as measured by wall clock time. In general, however, we use the number of GMRES iterations as a proxy for computational work, with each multi-stage GMRES iteration counting as s matrix application equivalents, and each single-stage GMRES iteration counting as one matrix application equivalent.

4.3 Structure Solvers

The ODE from the semi-discretization of the structure, found in equation 4.21, has the same form as that of the fluid, but our CG discretization and the domain itself mean there are some significant differences. Firstly, the physical domain has a smaller area, requiring fewer elements. In the case of rigid bodies, we only have two degrees of freedom. In larger structures, where we employ the neo-Hookean Elasticity model, the DG discretization used has no duplicate nodes on the edge of elements, reducing the number of degrees of freedom. For these applications, however, we still use the METIS software described in [43] to decompose the domain, and perform the discretization and matrix assembly in parallel. As with the fluid, stage parallelism is not utilized in solving the IRK systems, in order to maintain a consistent partition between all RK methods.

4.3.1 Explicit Parallel Structure Solver

In solving the explicit RK stages for the structure, we have the equation

$$M^s k_i^s = r^s \left(u_n^s + \Delta t w_i^s\right) \tag{4.29}$$

with w_i^s defined, as usual, by

$$w_i^s = \sum_{j=1}^{i-1} a_{ij} k_j^s. \tag{4.30}$$

This differs very little from equation 4.22. Again, the residual is computed at each element locally, then added to global residual by the *stamping method*, where overlapping values are added together. Since we use a CG method for the structure, the mass matrix is not block diagonal, as it is with the CG used on the fluid. We forego the use of iterative methods for linear systems and solve directly using the MUMPS multifrontal parallel sparse direct solver used in [30]. This software package factors and stores the mass matrix once, making parallel solves very efficient.

4.3.2 Implicit Parallel Structure Solvers

As with the fluid, the implicit stages for the structure come from implicit stages of EDIRK, ESDIRK, and DIRK methods, as well as IRK methods. We solve these using Newton's method, which results in our needing to solve the linear equation

$$\left(M^s - a_{ii}\Delta t \frac{dr^s}{du^s}\right)\delta k_i^s = r^s \tag{4.31}$$

for single stages and

$$\left(\boldsymbol{A}^{-1} \otimes M^s - \boldsymbol{J}\left(u_n^s + \Delta t \boldsymbol{W}^s\right)\right)\delta \boldsymbol{W}^s = \boldsymbol{R}^s\left(u_n^s + \Delta t \boldsymbol{W}^s\right) \tag{4.32}$$

for IRK methods, where multiple stages are computed at once. As with explicit solves, we use MUMPS to solve these systems in parallel.

In both single- and multi-stage cases, each process contributes a local portion of the matrix in coordinate form, supplying global coordinates and corresponding entries. The stamping method is used, with MUMPS completing assembly one element at a time, summing duplicate entries from the boundary nodes automatically. In both the single- and multi-stage cases, the structure of the matrix is fixed, so the global coordinates need only be supplied once. MUMPS then factors the matrix symbolically. This prefactorization is reused, leaving only the numerical factorization phase at each step. These direct solves are not particularly efficient, but, in practice, take significantly less time than fluid solves.

Chapter 5

Fluid-Structure Interaction

Having laid out the governing equations in chapter 2 for both structure and fluid, and the discretization of these equations in chapter 4, we focus on their interaction. So far, we have described the effects of the structure on the fluid in section 2.2 as a domain deformation, which we deal with via an ALE formulation. The fluid applies a traction force on the structure, described in equation 2.39 for the rigid body case and in equation 2.45 in the flexible neo-Hookean case.

5.1 Coupling

In order to facilitate ease of computation, we add two additional requirements to the structure and fluid spatial discretizations described in sections 4.1.1 and 4.1.2, namely that the elements use the same polynomial order p, and be edge-wise matching (face-wise in 3D) at the interface. Two elements $\mathcal{T}_1$ and $\mathcal{T}_2$ are edge-wise matching on an interface Γ_{12} between two meshes if they satisfy

$$\partial\mathcal{T}_1 \cap \Gamma_{12} = \partial\mathcal{T}_2 \cap \Gamma_{12}. \tag{5.1}$$

This is illustrated in figure 5.1.

This approach of requiring edge-wise matching meshes of the same polynomial order imposes some restrictions on the meshing process, as the fluid and structure meshes cannot be chosen independently. As the length of the elements tangent to the interface must be the same in both meshes, this leads to so-called mesh constraint leak in this direction. Once the meshes are formed, however, these restrictions ease simulation-time processes significantly. As the edge nodes on the interface line up, passing data across the boundary is very straightforward, accomplished with a static lookup table instead of interpolation. Additionally, the fluid mesh deformation due to structure movement is easy to construct. These advantages generally outweigh the difficulties caused in mesh generation.

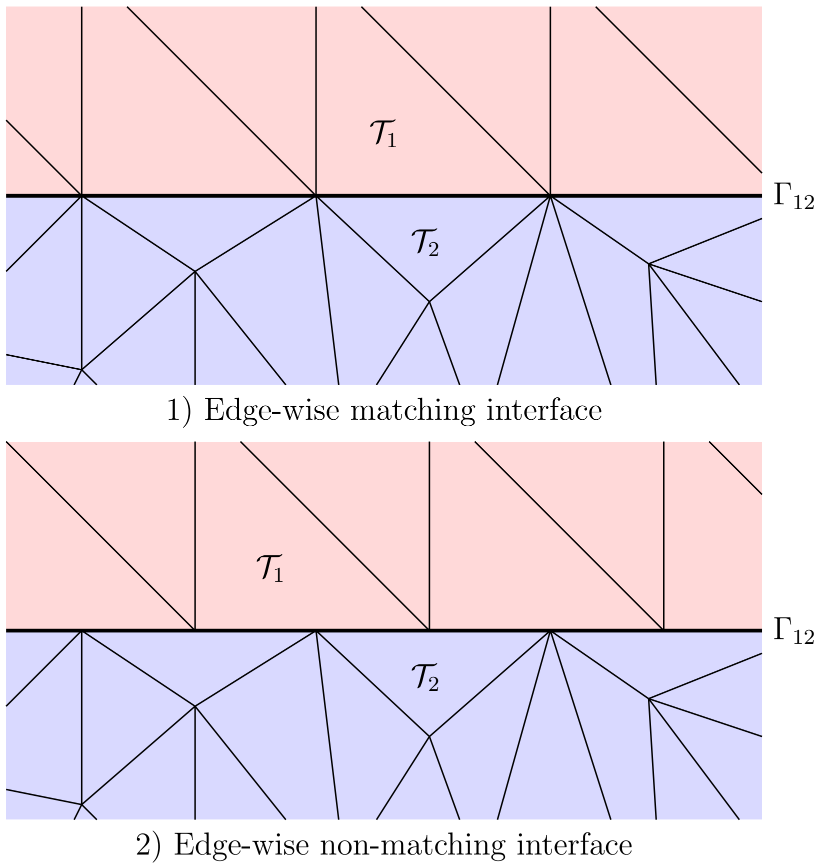


Figure 5.1: Two mesh interfaces illustrating a matching (1) and a non-matching (2) interface between a structured mesh (above in each, colored red) and an unstructured mesh (below in each, colored blue): The interface, Γ_{12}, is shown in bold.

5.1.1 Fluid-to-structure coupling

When dealing with flexible structures, described by the equations in 2.4, the fluid-to-structure coupling is described by a traction applied at the fluid-structure interface. With the meshes aligned, the traction is applied to the structure by computing the momentum flux through the interface with no interpolation. This is done pointwise at Gauss integration nodes instead of solution nodes to maximize accuracy.

For rigid bodies, however, some integration is required, as the total force is defined as

$$\boldsymbol{F} = \oint_{\partial V} \boldsymbol{t}\, dx \tag{5.2}$$

(or dA in three dimensions). This is computed numerically with Gaussian integration at each face e on the interface. The traction is computed as $\boldsymbol{t}\left(\boldsymbol{x}_i^{\{e\}}\right)$ at each Gauss node $\boldsymbol{x}_i^{\{e\}}$

with $1 \le i \le n_g$. Weighting and summing, we arrive at the linear combination

$$\boldsymbol{F} = \sum_{e \in \Gamma} \sum_{i=1}^{n_g} w_i^{\{e\}} \boldsymbol{t} \left(\boldsymbol{x}_i^{\{e\}} \right) \tag{5.3}$$

which is used for the numerical force. The torque about a point $\boldsymbol{x}_0$, usually the rotation axis or center of mass, is

$$\boldsymbol{\tau} = \oint_{\partial V} (\boldsymbol{x} - \boldsymbol{x}_0) \times \boldsymbol{t} \, dx \tag{5.4}$$

and is similarly calculated at the Gauss nodes as

$$\tau = \sum_{e \in \Gamma} \sum_{i=1}^{n_g} w_i^{\{e\}} \left(\boldsymbol{x}_i^{\{e\}} - \boldsymbol{x}_0 \right) \boldsymbol{t} \left(\boldsymbol{x}_i^{\{e\}} \right). \tag{5.5}$$

This is somewhat more complicated in the three dimensional case, which we do not consider here.

5.1.2 Structure-to-fluid coupling

As discussed in section 2.2, the effect of the structure movement on the fluid is a change in the fluid domain shape. The effect of this on the governing equations is dealt with using an ALE framework. The change in the shape of the computational domain of the fluid also requires the fluid mesh to move, however. For very small movement of the structure, moving the boundary nodes alone is sufficient, but if the magnitude of the movement of the structure approaches or exceeds the size of the fluid boundary nodes, the fluid mesh can become very ill conditioned or even inverted at the boundary. To avoid this, we move the interior nodes in addition to the boundary nodes. There are several ways to handle the necessary mesh deformations. For rigid body applications, we simply move the entire mesh with the structure, resulting in no relative motion within the fluid mesh. With flexible structures, this is not possible, and we adopt the method from [30] using *radial basis functions*, which is appropriate for the small to moderate deformations we observe.

The basic premise of our mesh deformation scheme is to establish a mapping from each position $\boldsymbol{X}$ in the reference fluid domain to a point $\boldsymbol{x}$ in the physical fluid domain, with a set of control points, $\{\boldsymbol{X}_j\}$ fixed according to the structure deformation, with

$$\boldsymbol{x}_j = \boldsymbol{x}\left(\boldsymbol{X}_j\right) \;\; \forall \; 1 \le j \le n. \tag{5.6}$$

We then define a set of radial basis functions ϕ_j, characteristic radii r_j and weights $\boldsymbol{\alpha}_j$ to construct the mapping

$$\boldsymbol{x}(\boldsymbol{X}) = \sum_{j=1}^{n} \boldsymbol{\alpha}_j \phi_j \left(\frac{\|\boldsymbol{X} - \boldsymbol{X}_j\|_2}{r_j} \right) + \boldsymbol{p}(\boldsymbol{X}) \tag{5.7}$$

where $\boldsymbol{p}$ is a linear polynomial, whose coefficients are determined by imposing equation 5.6 and the additional requirement that

$$\sum_{j=1}^{n} \boldsymbol{\alpha}_j \boldsymbol{q}\left(\boldsymbol{X}_j\right) = 0 \tag{5.8}$$

for all polynomials $\boldsymbol{q}$ of degree less than or equal to $\boldsymbol{p}$.

In order to compute the mapping efficiently, we precompute the matrix

$$M_{ij} = \phi\left(\frac{\|\boldsymbol{X}_j - \boldsymbol{X}_j\|_2}{r_j}\right) \tag{5.9}$$

and define $\varphi = [1, X]$ to convert the mapping into the linear system

$$\begin{bmatrix} M & \varphi \\ \varphi^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} \\ 0 \end{bmatrix} \tag{5.10}$$

whose matrix we pre-factor in parallel. This is generally faster than computing the factorization at simulation time, as the control points are the boundary nodes of the fluid, and thus significantly less numerous than the total number of nodes in the mesh. One last benefit of the radial basis function approach is that the mapping velocity $\boldsymbol{v}$ is readily available as

$$\boldsymbol{v}(\boldsymbol{X}) = \sum_{j=1}^{n} \frac{d\boldsymbol{\alpha}_j}{dt} \phi\left(\frac{\|\boldsymbol{X}_j - \boldsymbol{X}_j\|_2}{r_j}\right) + \frac{d\boldsymbol{p}}{dt}\left(\boldsymbol{X}\right) \tag{5.11}$$

and used in the ALE formulation described in section 2.2 directly. For more information on radial basis functions, see [10, 12, 30].

5.2 Diagonally Implicit Runge-Kutta Temporal Integration

With the discretization and coupling in place, we have our full combined system of ordinary differential equations to integrate in time, which we write as

$$M\frac{du}{dt} = r(u) \tag{5.12}$$

with

$$M = \begin{bmatrix} M^f & \\ & M^s \end{bmatrix}, u = \begin{bmatrix} u^f \\ u^s \end{bmatrix}, r(u) = \begin{bmatrix} r^f(u^f; x(u^s)) \\ r^s(u^s; t(u^f)) \end{bmatrix}.$$

We drop the bold notation for single-stage vectors of the ODE, to differentiate between the multi-stage vectors found in the full IRK time integration. The structure residual depends on the applied traction t and the fluid residual depends on the ALE mesh motion written as x. When using DIRK schemes to integrate in time, we use the method from [78, 30, 37]. This method relies on the key fact that the structure residual can be broken into two parts:

$$r^s(u^s; t(u^f)) = r^{ss}(r^s) + r^{sf}(t(u^f)) \tag{5.13}$$

which separates out the fluid effect from the rest of the function. We see from section 5.1.1 that the structure residual is linear in t, which allows us to write this as

$$r^s(u^s; t(u^f)) = r^s(u^s; \tilde{t}) + r^{sf}(t(u^f) - \tilde{t}) \tag{5.14}$$

for any arbitrary traction $\tilde{t}$. From this property arises the split formulation,

$$M\frac{du}{dt} = \left[\begin{array}{c} r^f(u^f; x(u^s)) \\ r^s(u^s; \tilde{t}) \end{array} \right] + \left[\begin{array}{c} \\ r^{sf}(t(u^f) - \tilde{t}) \end{array} \right]. \tag{5.15}$$

The use of ERK methods for this ODE is straightforward, but directly-applied DIRK methods require the computation of the Jacobian of the residual,

$$J = \left[\begin{array}{cc} J^{ff} & J^{fs} \\ J^{sf} & J^{ss} \end{array} \right]. \tag{5.16}$$

Though the diagonal blocks of equation (5.16) are readily available in the single-physics solvers we use, 3DG, the off-diagonal blocks are not. Thus, we avoid calculating them, using the method from [30]. We use an ESDIRK method, where a prediction, structure solve, fluid solve, structure correction sequence is followed at each implicit stage. The final structure correction is calculated explicitly, whereas the structure and fluid solves are implicit, using only diagonal blocks of the Jacobian in equation (5.16). The ESDIRK is paired with an ERK, as described in section 3.5. The traction prediction, $\tilde{t}$, is calculated at each implicit stage by a predictor suggested by van Zuijlen in [78]:

$$\tilde{t}_i = \sum_{j=1}^{i-1} \frac{\hat{a}_{ij} - a_{ij}}{a_{ii}} t_j \tag{5.17}$$

where a_{ij} come from the butcher array of the ESDIRK, and $\hat{a}_{ij}$ are from the paired ERK. The t_j are the calculated values of the traction at each stage. This method of time integration is described in algorithm 3, with Gauss-Seidel iterations.

It should be noted that algorithm 3 is not strictly an IMEX scheme as defined in algorithm 1, as the explicit term r^{sf} is not actually calculated separately, but only as a part of r^s in the final stage correction after the Gauss-Seidel iterations. It has been found in [30] that these Gauss-Seidel iterations are not necessary to reach the design accuracy of the ESDIRK method used, but can improve the stability of the method.

Algorithm 3: ESDIRK time integration scheme for coupled fluid-structure system

```
Result: ODE Solution at time $t_0 + \Delta t$, $u^f_{n+1}$ and $u^s_{n+1}$
Input: Paired ESDIRK $A$, $b$, $c$ and ERK $\hat{A}$, $\hat{b}$, $\hat{c}$ and initial values $u^f_n$, $u^s_n$
$t_1 \leftarrow t(u^f_n)$;
Solve $M^s k^s_1 = r^s(u^s_n; t_1)$ for $k^s_1$;
Solve $M^f k^f_1 = r^f(u^f_n; x(u^s_n))$ for $k^f_1$;
for $i$ from 2 to $s$ do
    $\tilde{t}_i \leftarrow \sum_{j=1}^{i-1} \frac{\hat{a}_{ij}-a_{ij}}{a_{ii}} t_j$;
    for $j$ from 1 to number of Gauss-Seidel iterations do
        Solve $M^s k^s_i = r^s(u^s_n + \Delta t w^s_i; \tilde{t}_i)$ for $k^s_i$;
        Solve $M^f k^f_i = r^f(u^f_n + \Delta t w^f_i; x(u^s_n + \Delta t w^s_i))$ for $k^f_i$;
        $\tilde{t}_i \leftarrow t(u^f_i + \Delta t w^f_i)$;
    end
    $t_i \leftarrow t(u^f_i + \Delta t w^f_i)$;
    Solve $M^s k^s_i = r^s(u^s_n + \Delta t w^s_i; t_i)$ for $k^s_i$;
end
$u^s_{n+1} \leftarrow u^s_n + \Delta t \sum_{i=1}^{s} b_i k^s_i$;
$u^f_{n+1} \leftarrow u^f_n + \Delta t \sum_{i=1}^{s} b_i k^f_i$;
# Note that $w_i$ is defined in equation 3.13
```

5.3 Fully Implicit Runge-Kutta Temporal Integration

We now develop a time integration scheme for fully implicit Runge-Kutta methods called Corrected Implicit Runge-Kutta (CIRK). Our method takes advantage of the property of linearity in the traction (equation 5.14) as the ESDIRK method shown in algorithm 3, and is similarly fully partitioned, that is, it can be implemented with separate fluid and structure solvers, allowing for reuse of existing software components. The key difference between these time integration methods is that while the ESDIRK method has stages computed in sequence, with opportunity for traction prediction, our CIRK method does not present this opportunity, as all stages are computed at once. This creates a need for the development of prediction methods, two of which we present here.

5.3.1 Prediction-based IRK Solver

Our method, shown in Algorithm 4, uses predicted stage values $\tilde{\boldsymbol{W}}^f$ and $\tilde{\boldsymbol{W}}^s$, as approximations of those defined in equation 3.20. While only $\tilde{\boldsymbol{W}}^f$ is strictly necessary, both are useful in practice as starting values for the Newton-iteration in each non-linear system. The algorithm follows the general pattern found in the method developed in [30], with traction predictions preceding two non-linear solves, followed by a structure correction.

Algorithm 4: Corrected IRK Method

```
Result: ODE Solution at time $t_0 + \Delta t$, $\boldsymbol{U}^f_{n+1}$ and $\boldsymbol{U}^s_{n+1}$
Input: IRK method $A$, $b$, $c$, initial values $\boldsymbol{U}^f_n$, $\boldsymbol{U}^s_n$, & predicted values $\tilde{\boldsymbol{W}}^f$, $\tilde{\boldsymbol{W}}^s$
$\boldsymbol{W}^f \leftarrow \tilde{\boldsymbol{W}}^f$;
for i from 1 to number of Gauss-Seidel iterations do
    $\tilde{\boldsymbol{T}} \leftarrow \boldsymbol{T}\left(\boldsymbol{U}^f_n + \Delta t \boldsymbol{W}^f\right)$;
    Solve $\left(\boldsymbol{A}^{-1} \otimes \boldsymbol{M}^s\right) \tilde{\boldsymbol{W}}^s = \boldsymbol{R}^s\left(\boldsymbol{U}^s_n + \Delta t \tilde{\boldsymbol{W}}^s; \tilde{\boldsymbol{T}}\right)$ for $\tilde{\boldsymbol{W}}^s$;
    Solve $\left(\boldsymbol{A}^{-1} \otimes \boldsymbol{M}^f\right) \boldsymbol{W}^f = \boldsymbol{R}^f\left(\boldsymbol{U}^f_n + \Delta t \boldsymbol{W}^f; \boldsymbol{X}\left(\boldsymbol{U}^s_n + \Delta t \tilde{\boldsymbol{W}}^s\right)\right)$ for $\boldsymbol{W}^f$;
end
Solve $\left(\left(\boldsymbol{A}^{-1} \otimes \boldsymbol{M}^s\right) - \Delta t \boldsymbol{J}^{ss}\right) \Delta \boldsymbol{W}^s = \boldsymbol{R}^{sf}\left(\boldsymbol{T}\left(\boldsymbol{U}^f_n + \Delta t \boldsymbol{W}^f\right) - \tilde{\boldsymbol{T}}\right)$ for $\Delta \boldsymbol{W}^s$ where
  $\boldsymbol{J}^{ss}$ is recovered from last Newton iteration of most recent Structure Solve;
$\boldsymbol{W}^s \leftarrow \tilde{\boldsymbol{W}}^s + \Delta \boldsymbol{W}^s$;
$\boldsymbol{U}^s_1 = \boldsymbol{U}^s_0 + \Delta t \left(b^T \boldsymbol{A}^{-1} \otimes I\right) \boldsymbol{W}^s$;
$\boldsymbol{U}^f_1 = \boldsymbol{U}^f_0 + \Delta t \left(b^T \boldsymbol{A}^{-1} \otimes I\right) \boldsymbol{W}^f$;
```

The correction step in algorithm 4 relies on the linear property from equation 5.15, but differs from the correction step in its use of the Jacobian $\boldsymbol{J}^{ss}$. The correction step is derived from the formulation in equation 5.15, where the structure portion is

$$M^s \frac{du^s}{dt} = r^s(u^s; \tilde{t}) + r^{sf}(t(u^f) - \tilde{t}). \tag{5.18}$$

Under the transformation laid out in 3.20, we have

$$\left(\boldsymbol{A}^{-1} \otimes \boldsymbol{M}^s\right) \boldsymbol{W}^s = \boldsymbol{R}^s\left(\boldsymbol{U}^s_n + \Delta t \boldsymbol{W}^s; \tilde{T}\right) + \boldsymbol{R}^{sf}\left(T\left(\boldsymbol{U}^f_n + \Delta t \boldsymbol{W}^f\right) - \tilde{T}\right). \tag{5.19}$$

Let $\tilde{\boldsymbol{W}}^s$ be a solution to the structure ODE with predicted traction $\tilde{T}$,

$$\left(\boldsymbol{A}^{-1} \otimes \boldsymbol{M}^s\right) \tilde{\boldsymbol{W}}^s = \boldsymbol{R}^s\left(\boldsymbol{U}^s_n + \Delta t \tilde{\boldsymbol{W}}^s; \tilde{T}\right), \tag{5.20}$$

and let the difference be defined as $\Delta \boldsymbol{W}^s := \boldsymbol{W}^s - \tilde{\boldsymbol{W}}^s$. Linearizing about $\boldsymbol{U}^s_n + \Delta t \tilde{\boldsymbol{W}}^s$, we have

$$\boldsymbol{R}^s\left(\boldsymbol{U}^s_n + \Delta t \tilde{\boldsymbol{W}}^s + \Delta t \Delta \boldsymbol{W}^s; \tilde{T}\right) = \boldsymbol{R}^s\left(\boldsymbol{U}^s_n + \Delta t \tilde{\boldsymbol{W}}^s; \tilde{T}\right) + \boldsymbol{J}^{ss} \Delta t \Delta \boldsymbol{W}^s + \boldsymbol{\epsilon}$$

where

$$\|\boldsymbol{\epsilon}\| = \mathcal{O}\left(\|\Delta t \Delta \boldsymbol{W}^s\|^2\right) = \mathcal{O}\left(\Delta t^2 \|\Delta \boldsymbol{W}^s\|^2\right).$$

This is equivalent to

$$\boldsymbol{R}^s\left(\boldsymbol{U}^s_n + \Delta t \boldsymbol{W}^s; \tilde{T}\right) - \boldsymbol{R}^s\left(\boldsymbol{U}^s_n + \Delta t \tilde{\boldsymbol{W}}^s; \tilde{T}\right) = \boldsymbol{J}^{ss} \Delta t \Delta \boldsymbol{W}^s + \boldsymbol{\epsilon},$$

meaning that the difference of equations 5.19 and 5.20 can be expressed as

$$\left(\boldsymbol{A}^{-1} \otimes \boldsymbol{M}^s\right) \Delta \boldsymbol{W}^s = \Delta t \boldsymbol{J}^{ss} \Delta \boldsymbol{W}^s + \boldsymbol{R}^{sf} \left(T\left(\boldsymbol{U}_0^f + \Delta t \boldsymbol{W}^f\right) - \tilde{T}\right) + \boldsymbol{\epsilon}. \tag{5.21}$$

For small enough $\Delta \boldsymbol{W}^s$, the higher order term, $\boldsymbol{\epsilon}$, can be dropped, giving the structure correction step of algorithm 4. Note that because the structure solve is performed with Newton iterations, the Jacobian, $\boldsymbol{J}^{ss}$, will have already been calculated, incurring only the extra cost of one solve.

The CIRK method (algorithm 4) allows for the implementation of additional Gauss-Seidel subiterations even more naturally than the comparison ESDIRK method from [30]. Whereas the ESDIRK method takes only a predicted traction, but not predicted stage values, our CIRK method relies on predicted stage values, giving a natural starting value for Newton's method. The stage values from the previous Gauss-Seidel subiteration naturally replace these predicted stage values after the first subiteration. Though it is possible to use a similar strategy with an ESDIRK method, we maintain the original implementation as described in [30] when using subiterations. In both methods, these subiterations improve the stability of the method, but are not required to achieve the design order of the RK method used. When using them, however, we have found that the solution converges to the monolithic case after a sufficient number of subiterations. This number is investigated further in section 6.2. In general, however, we hold the number of Gauss-Seidel iterations at one, executing the for-loop only once, unless otherwise stated.

It should be noted that both the cost and accuracy of the CIRK method is highly dependent on the accuracy of the predicted values $\tilde{\boldsymbol{W}}^f$ and $\tilde{\boldsymbol{W}}^s$. Good predicted values reduce the number of Newton iterations by offering a good starting iteration. Additionally, the value of $\tilde{\boldsymbol{W}}^f$ determines the traction at each stage, which effects the accuracy of the method.

5.3.2 Stage-Value Predictors

We propose two predictor algorithms for the Corrected IRK Method presented in section 5.3.1. The first, described in algorithm 5, involves explicit timestepping from time $t_0 + c_{i-1}\Delta t$ to $t_0 + c_i \Delta t$ for each c_i in the vector c of the IRK butcher array, with c_0 defined as 0. The advantage of the ERK prediction of the stage values is that it involves no non-linear solves, reducing computational work of prediction to s times the cost of a step with the chosen ERK. For stiff problems, however, explicit prediction limits the size of the timestep. We find that in some cases that the Corrected IRK method (algorithm 4), paired with the ERK prediction from algorithm 5, faces similar limits on the size of Δt to a simple application of RK4, the fourth-order ERK from equation 3.16, though this is highly problem-dependent.

The second prediction method we present is based on implementation of a modified version of the IMEX algorithm presented in [30] and discussed in section 5.2, which relies on a paired EDIRK and ERK. Our method uses stage-values to build $\tilde{\boldsymbol{W}}^f$ and $\tilde{\boldsymbol{W}}^s$ instead

Algorithm 5: ERK Prediction Algorithm

```
Result: Predicted Stage Values $\tilde{\boldsymbol{W}}^f$, $\tilde{\boldsymbol{W}}^s$
Input: $c = [c_1, ..., c_s]$ from Butcher array of IRK
$\boldsymbol{A}$, $b$ from Butcher array of $r$-stage ERK
Initial values $u_n^f$, $u_n^s$
$c_0 \leftarrow 0$;
$u_n \leftarrow \begin{bmatrix} u_0^f \\ u_0^s \end{bmatrix}$;
$\tilde{\boldsymbol{W}}^f \leftarrow 0$;
$\tilde{\boldsymbol{W}}^s \leftarrow 0$;
for $i = 1$ to $s$ do
    for $j = 1$ to $r$ do
        Solve $Mk_j = r\left(u_n^f + (c_i - c_{i-1})\Delta t \sum_{p=1}^{j-1} A_{ip}k_p\right)$ for $k_j$;
    end
    $d_i \leftarrow (c_i - c_{i-1}) \sum_{p=1}^{j-1} b_p k_p$;
    if $i = 1$ then
        $\tilde{W}_i^f \leftarrow d_i^f$;
        $\tilde{W}_i^s \leftarrow d_i^s$;
    end
    if $i > 1$ then
        $\tilde{W}_i^f \leftarrow \tilde{W}_{i-1}^f + d_i^f$;
        $\tilde{W}_i^s \leftarrow \tilde{W}_{i-1}^s + d_i^s$;
    end
    $u_n^f \leftarrow u_n^f + \Delta t d_i^f$;
    $u_n^s \leftarrow u_n^s + \Delta t d_i^s$;
end
# Note that $W_i$ is the $i^{\text{th}}$-stage component of $\boldsymbol{W}$
```

of updating the solution u_n. We require an EDIRK-ERK pair which is chosen to have the same c vector in their butcher arrays as the IRK, adding an initial zero as necessary to allow for consistency in the EDIRK scheme. We have developed just such a pair for several IRK methods in section 3.6. The EDIRK prediction method is shown in algorithm 6.

Algorithm 6: DIRK Prediction Algorithm

```
Result: Predicted Stage Values $\tilde{\boldsymbol{W}}^f$, $\tilde{\boldsymbol{W}}^s$
Input: Paired EDIRK $A$, $b$, $c$ and ERK $\hat{A}$, $\hat{b}$, $\hat{c}$ and initial values $u_n^f$, $u_n^s$
Solve $M^f k_1^f = r^f(u_n^f; x(u_n^s))$ for $k_1^f$;
Solve $M^s k_1^s = r^s(u_n^s; t(u_n^f))$ for $k_1^s$;
for $i = 2$ to $s$ do
    $\tilde{t} \leftarrow \sum_{j=1}^{i-1} \frac{\hat{a}_{ij} - a_{ij}}{a_{ii}} t_j$;
    Solve $M^s k_i^s = r^s\left(u_n^s + \Delta t w_j^s; \tilde{t}\right)$ for $k_i^s$;
    Solve $M^f k_i^f = r^f\left(u_n^f + \Delta t w_j^f; x\left(u_n^s + \Delta t w_j^s\right)\right)$ for $k_i^f$;
    $t_i \leftarrow t\left(u_n^f + \Delta t w_j^f\right)$;
    Solve $M^s k_i^s = r^s(u_n^s + \Delta t w_i^s; t_i)$ for $k_i^s$;
    $\tilde{W}_i^f \leftarrow \sum_{j=1}^{i} a_{ij} k_j^f$;
    $\tilde{W}_i^s \leftarrow \sum_{j=1}^{i} a_{ij} k_j^s$;
end
# Note that $w_i$ is defined in equation 3.13
```

Though this prediction algorithm requires solving one non-linear system of size N for each stage of the IRK method, for a total of s solves, the added stability of using an IMEX method allows for much larger timesteps than with explicit prediction, though at a higher cost per step. However, with the use of an IRK, a non-linear system of size $N \cdot s$ will ultimately need to be solved, whose cost contributes a large portion of the total amount of work needed. Furthermore, the s solves give more accurate predictions for the $\boldsymbol{W}$ vectors used in algorithm 4, offering a starting-value for Newton iterations, thus lowering the cost of solving the larger system.

5.4 Verification

5.4.1 One-Dimensional Piston

We first apply our CIRK method to a gas-filled cylinder with one weighted spring-moderated free-moving piston and one fixed wall (see figure 5.2). The gas is modeled by the Euler equations for inviscid flow, and the movement of the piston is governed by Hooke's law:

$$m\frac{\partial^2 y}{\partial t^2} = k(y_0 - y) + p\left(u_f\right) \tag{5.22}$$

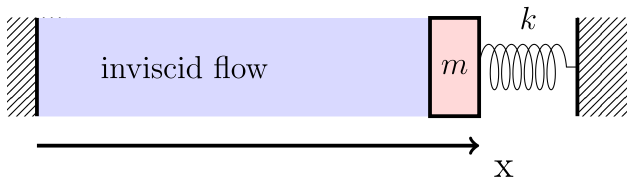


Figure 5.2: One-dimensional piston system (image from [37])

where m and y are the mass and position of the piston, respectively, k is the spring stiffness, y_0 is the rest position of the piston, u_f is the state of the gas at the piston face, and the pressure p is defined by

$$p = (\gamma - 1)\left(\rho E - \frac{\rho v^2}{2}\right).$$

Equation 5.22 is converted to an ordinary differential equation in two variables, y and $\frac{\partial y}{\partial t}$, where

$$u^s = \begin{bmatrix} y \\ \frac{\partial y}{\partial t} \end{bmatrix}$$

and a mesh of n equally spaced elements is imposed on the fluid domain, with ALE mesh motion described by equal element sizes, giving a linear relationship between x-coordinate and mesh velocity. Together these ordinary differential equations for a system of the form found in equation 5.12. For our implementation, we choose $y_0 = 1$, $\gamma = 1.4$, $n = 10$ and $p = 4$. A spring stiffness of $k = 1000$ is used, which makes the problem unsuitable for explicit methods.

The Corrected IRK method from algorithm 4, using third- and fifth-order IRK methods from equation 3.26 and a sixth-order IRK from equation 3.27, is implemented to solve the piston problem using both the explicit prediction from algorithm 5 and the EDIRK prediction from algorithm 6 (hereafter called ERK-CIRK and EDIRK-CIRK). We perform a convergence analysis of these methods and the ESDIRK method from [30] (algorithm 3 with $m = 1$), with results shown in the left panels of figures 5.3, 5.4, and 5.5. With the sixth-order method, we employ two explicit predictors, one using the fourth-order ERK from equation 3.16, and the other using the fifth-order ERK

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & 0 & 0 & 0 & 0 \\
\frac{3}{5} & \frac{3}{10} & -\frac{9}{10} & \frac{6}{5} & 0 & 0 & 0 \\
1 & -\frac{11}{54} & \frac{5}{2} & -\frac{70}{27} & \frac{35}{27} & 0 & 0 \\
\frac{7}{8} & \frac{1631}{55296} & \frac{175}{512} & \frac{575}{13824} & \frac{44275}{110592} & \frac{253}{4096} & 0 \\
\hline
 & \frac{37}{378} & 0 & \frac{250}{621} & \frac{125}{594} & 0 & \frac{512}{1771}
\end{array} \tag{5.23}$$

described in [21]. We compute a reference solution using the standard fourth-order ERK from [46] for calculating error. Though explicit prediction is not possible for larger timesteps, the order of the method is achieved in all six cases.

In order to make a fair comparison between the ESDIRK and CIRK methods, we adopt the same work metric as [56], discussed in depth in section 4.2.4, and compute the total number of n-by-n matrix-vector multiplications required across all stages, referring to this quantity as *matrix application equivalents*. As there are only two degrees of freedom in the structure, the cost of the non-linear fluid solves dominates, and the cost of structure solves is not counted. We compare the cost of solving the piston problem with the corrected IRK method with the ESDIRK method described in algorithm 3 in the right panels of figures 5.3 and 5.4. Again we use the parameter $m = 1$ and count all matrix application equivalents from $t_0 = 0$ to $t = 1.0$. The system is initialized with the piston at rest and in equilibrium with the fluid, which has velocity zero as both ends of the cylinder.

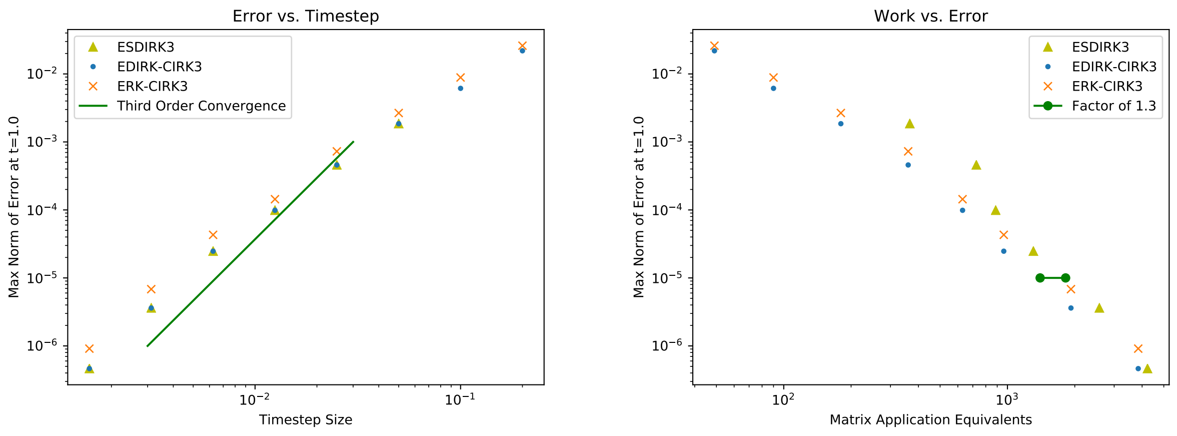


Figure 5.3: Third Order Method Comparison, Piston

We see clearly that both the three-stage fifth-order Radau IIA method and the two-stage third-order Radau IIA method achieve their design order of accuracy, at a comparable cost to the ESDIRK3 and ESDIRK5 methods used for comparison. This holds true using both the explicit prediction (ERK-CIRK) and implicit prediction (EDIRK-CIRK). Though there is, in the fifth-order case, a slightly lower cost for the explicit prediction, its use restricts the choice of timestep in this problem considerably. The four-stage sixth-order Lobatto IIIC method achieves its design order of accuracy with implicit prediction. We find that the fifth-order explicit predictor allows us to achieve the design-order of the sixth-order method through the smallest timesteps, whereas the fourth-order explicit predictor falls off the design order of the method for small timesteps.

In figure 5.7, we show the timesteps for which each method successfully solves the piston

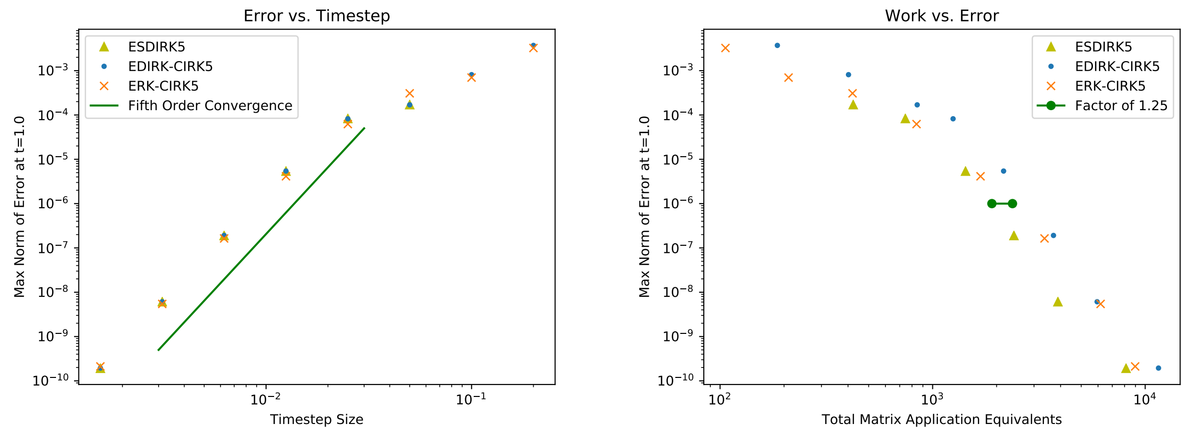


Figure 5.4: Fifth Order Method Comparison, Piston

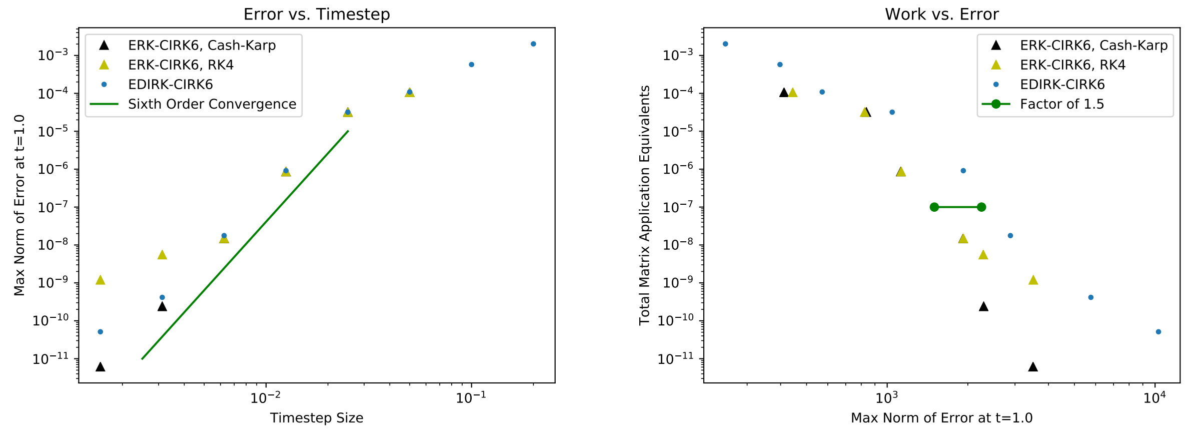


Figure 5.5: Sixth Order Method, Piston, with fifth-order Cash-Karp ERK predictor and fourth-order RK4 predictor

problem, varying the ratio of piston mass to initial density of the fluid at the piston face. Since our initial value for the fluid density at both ends of the cylinder is one, the mass ratio is equal to the mass of the piston. As the mass decreases, the sensitivity of the structure to the pressure increases, as is clear from equation 5.22. This decreases the maximum timestep size for some methods. Whereas the explicit methods are limited to smaller timesteps, especially at lower mass ratios, the ESDIRK method we use for comparison can be used with large timesteps. We find that the EDIRK-CIRK implementation possesses the same advantage over the ERK as the ESDIRK implementation does.

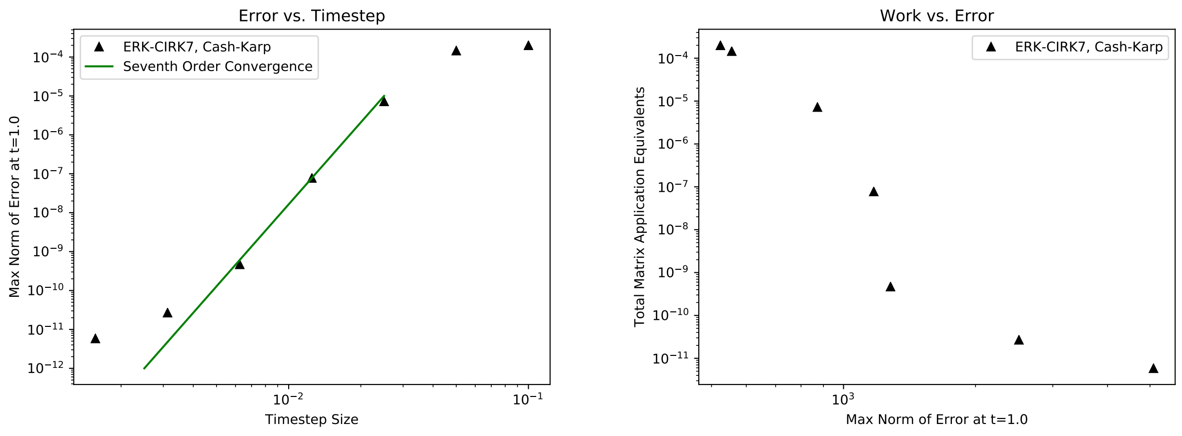


Figure 5.6: Seventh Order Method, Piston, with fifth-order Cash-Karp ERK predictor

5.4.2 Pitching and Heaving Airfoil

We next apply our method to a common test problem that is a two dimensional cross section of a wing with air flowing past it. Our simulation is comprised of a pitching and heaving NACA 0012 airfoil in viscous fluid with Reynolds number 1000. The far-field boundary conditions are set to horizontal velocity one and density one, and the boundary conditions on the airfoil interface are no-slip. We treat the airfoil as a rigid body, as set out in section 2.3, whose motion is described by two variables, vertical position and pitch. We choose the pitch as the free variable, with a prescribed vertical motion of the pivot point, $y(t)$, which satisfies $y(0) = 0$ and $y(1) = \frac{1}{4}$. This leaves two free variables describing the airfoil motion, which are θ and ω, the angle and angular velocity. The distance from the pivot to the center of mass is l, as shown in figure 5.8. For our simulation, the angle θ is governed by a torsional restoring force with torsional spring constant k and a torque τ resulting from the fluid. Setting the moment of inertia to I and mass to m, the governing equations are

$$\frac{\partial\theta}{\partial t} = \omega \tag{5.24}$$

$$I\frac{\partial\omega}{\partial t} = -k\theta - lm\cos(\theta)y''(t) - \tau. \tag{5.25}$$

The mesh moves as a rigid body fixed to the airfoil, so that the motion of the airfoil exactly determines the mesh motion. We apply our DG method from section 4.1.1 on a mesh with 1148 elements, and polynomial degree $p = 3$, yielding a total of 30,996 degrees of freedom. For our simulation, we choose parameters $I = 1$, $k = 0.1$, $l = 0.2$, $m = 1.0$, and a distance of $\frac{1}{3}$ from the leading edge of the airfoil to the pivot.

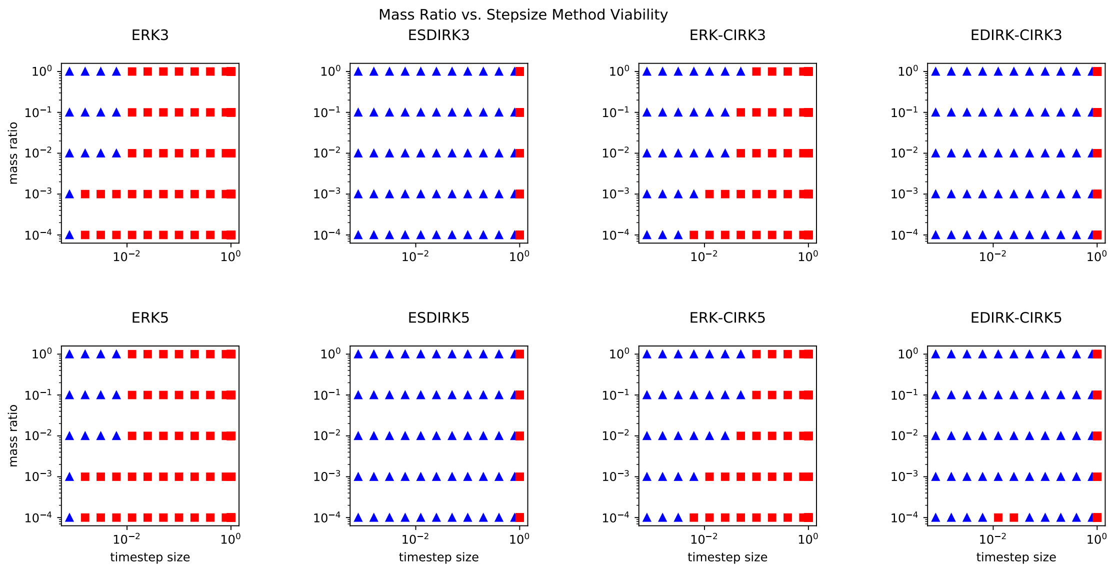


Figure 5.7: Mass Ratio Method Viability Comparison, Piston (ERK3 and ERK5 are Ralston's third-order method and fifth-order Cash-Karp, respectively. Blue triangles denote successful completion to time $t = 1.0$. Red squares denote failure of method to reach $t = 1.0$.)

The fluid is governed by the two-dimensional isentropic Navier-Stokes equations, using the simplification found in equation 2.11. The starting values for the fluid and the structure are chosen such that the structure is at rest and horizontal ($\theta = 0$) with a steady-state solution of the fluid. Error is measured in the fluid at time $t = 1.0$, with the max-norm, using a benchmark of the solution as determined by the ESDIRK5 scheme, with a timestep of $\Delta t = 1 \times 10^{-4}$. The structure equations, having only two degrees of freedom, are solved to machine precision, while we use the Newton-Krylov methods from section 4.2, with a relative GMRES tolerance of 10^{-5}, and an absolute Newton tolerance of 10^{-7}, for structure solves in both prediction EDIRK and final IRK.

The system is found to be too stiff to implement the explicit prediction described in algorithm 5 (ERK-CIRK) with a reasonable timestep size. Thus, we only use the EDIRK-CIRK method. We again apply this method using the fifth- and third-order RK methods from equation 3.26, and the sixth-order method from equation 3.27, with appropriate EDIRK-ERK pairs, as developed in section 3.6. For reference, we include the first-order methods from equation 3.43. Both the ESDIRK and EDIRK-CIRK methods exhibit the design order error convergence, in all four implementations, at comparable cost, where comparison is possible.

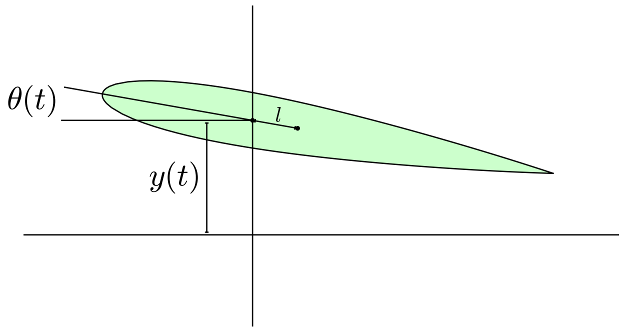


Figure 5.8: Schematic drawing of pitching airfoil.

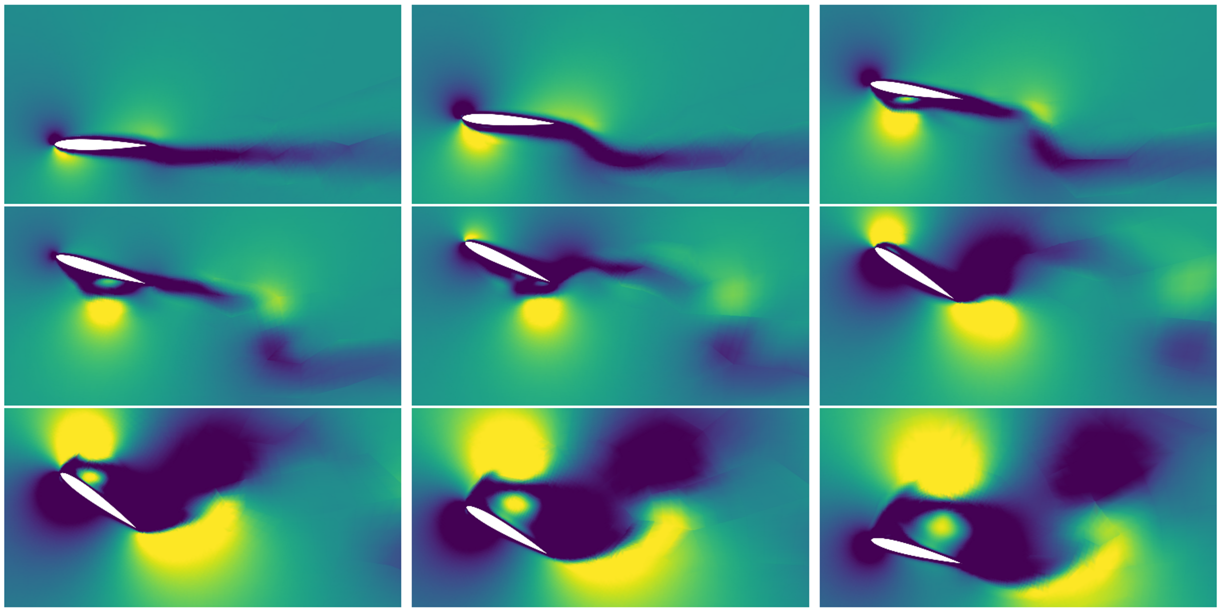


Figure 5.9: Solutions at $t \in \{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5\}$ of the pitching airfoil problem, with sinusoidal heaving. The color denotes Mach number.

In figure 5.14, we include the results of a study comparing the EDIRK-CIRK method with the reference-implementation ESDIRK in their abilities to successfully solve the airfoil problem to at least time $t = 1.0$, varying the timestep and airfoil mass. The moment of inertia, in order to stay proportional to the mass of the airfoil, is set equal to the mass in each case. The mass ratio is defined as the ratio of the mass of the structure to the mass of the fluid displaced. We again use the initial fluid density, one, so the mass ratio is mass of the airfoil divided by the area, 0.08221. Our results indicate that our Corrected IRK

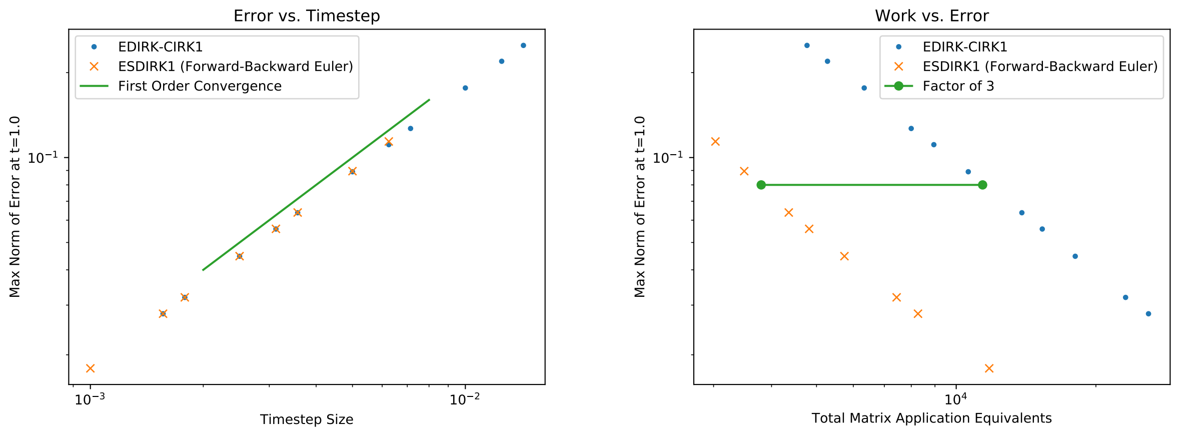


Figure 5.10: First Order Method Comparison, Airfoil

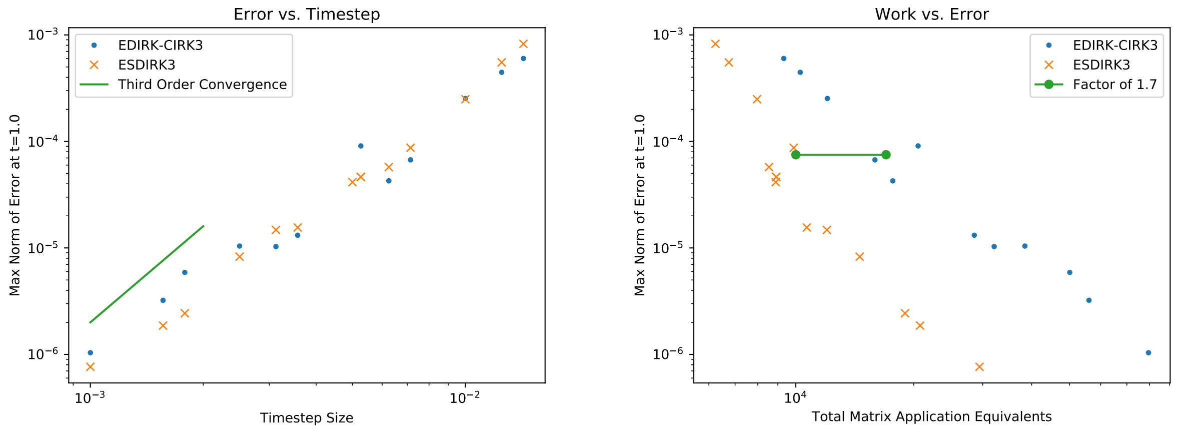


Figure 5.11: Third Order Method Comparison, Airfoil

method is stable over a wide range of values of timestep sizes (Δt) and mass ratios, where comparable ESDIRK methods are unstable.

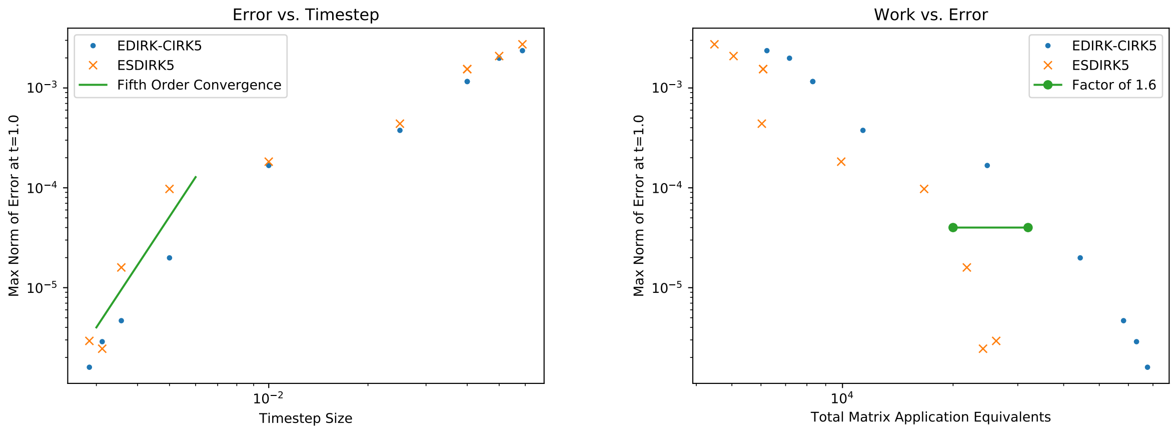


Figure 5.12: Fifth Order Method Comparison, Airfoil

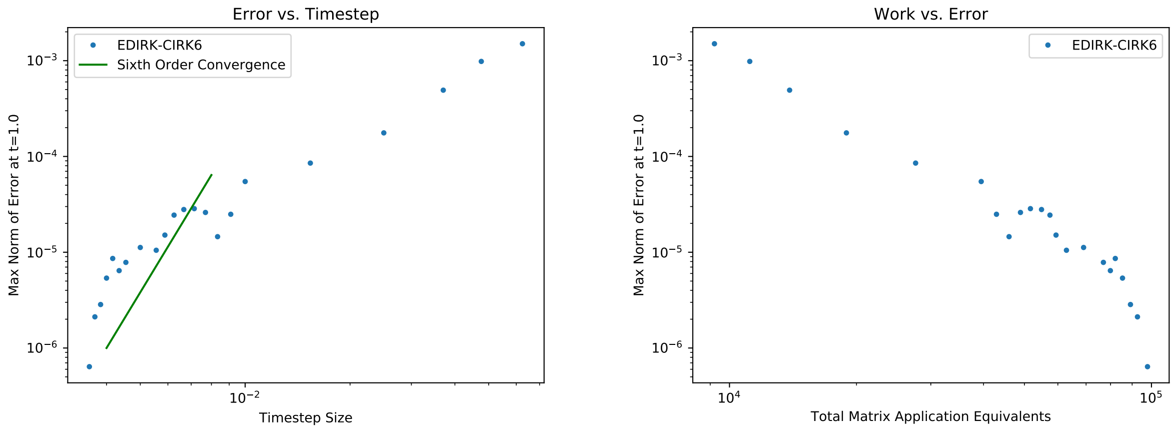


Figure 5.13: Sixth Order Method, Airfoil

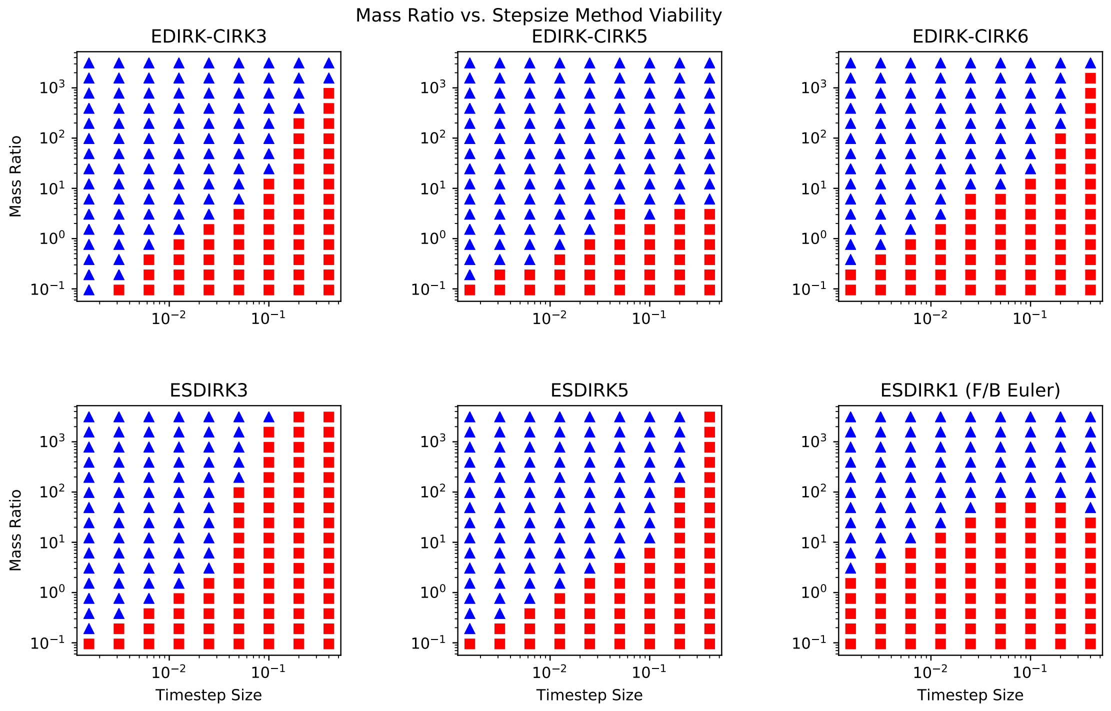


Figure 5.14: Mass Ratio Method Viability Comparison, Airfoil. Blue Triangles denote successful completion to time $t = 1.0$. Red squares denote failure of method to reach $t = 1.0$. Forward-Backward Euler is included for reference.

Chapter 6

Applications

6.1 Cantilever Behind Rigid Square Body

We now apply our method to a standard fluid-structure interaction test problem consisting of a flexible cantilever fixed to the back of a rigid square body. As discussed in [24], there are two standard variations of this problem, which differ in structure characteristics and fluid input velocity, but not geometry or viscosity. The first, Wall's Cantilever, was introduced in [62], and the second, Hübner's Cantilever, first appeared in [38]. We model the flexible cantilever with the neo-Hookean formulation from section 2.4. The two sets of parameters used are shown in Table 6.1.

Our fluid mesh contains 3,118 degree 3 elements, totaling 31,180 nodes and 124,720 degrees of freedom. We again apply the discontinuous Galerkin method described in section 4.1.1 to the fluid, and use the radial basis functions from section 5.1.2 to handle the mesh motion. The fixed square body and flexible cantilever are assigned no-slip boundary conditions, and the outer walls have far-field conditions corresponding to a uniform rightward input fluid velocity, as determined by table 6.1. The geometry is shown in figure 6.1. The flexible cantilever structure is modeled using the continuous Galerkin framework from section 4.1.2, with 34 degree $p = 3$ triangular elements, totalling 1,380 degrees of freedom. We use the Newton-Krylov methods described in 4.2. Our error tolerance for Newton's method is 10^{-8}, measured in the max norm, for all non-linear solves from ESDIRK, EDIRK prediction, and IRK methods. Our GMRES iterations are performed to a relative tolerance of 10^{-5}, also using the max norm.

Unlike in the airfoil and piston applications, this problem is found to be stiff enough that we increase the number of Gauss-Seidel iterations in algorithm 4 from one to two. We also perform two Gauss-Seidel iterations at each stage of the comparison ESDIRK methods, just as [30] does for this problem, and without observing the problems laid out in [42]. This change increases the stability of our method, allowing for reasonably large timesteps (see Table 6.2). No attempt at explicit prediction was made in this application. We observe von

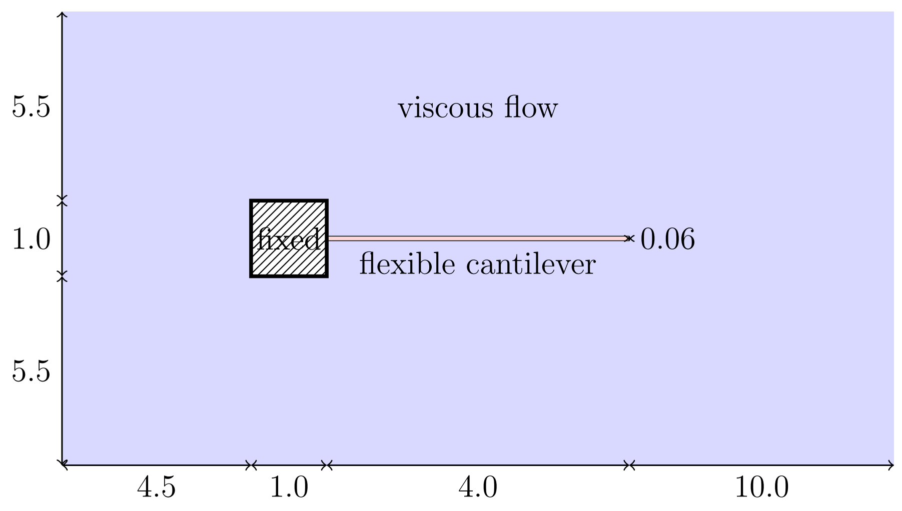


Figure 6.1: Two-dimensional Cantilever System (dimensions in cm, fluid boundary not to scale)

Parameter	Wall	Hübner
Input Fluid Velocity	$0.513\frac{\text{m}}{\text{s}}$	$0.315\frac{\text{m}}{\text{s}}$
Young modulus, E	$2.5 \times 10^6 \frac{\text{kg}}{\text{m s}^2}$	$0.2 \times 10^6 \frac{\text{kg}}{\text{m s}^2}$
Structure Density, ρ_s	$100\frac{\text{kg}}{\text{m}^3}$	$2000\frac{\text{kg}}{\text{m}^3}$
Poisson Coefficient, ν	0.35	0.35
Fluid Viscosity, μ_f	$1.82 \times 10^{-5} \frac{\text{kg}}{\text{m s}}$	$1.82 \times 10^{-5} \frac{\text{kg}}{\text{m s}}$
Fluid Density, ρ_f	$1.18\frac{\text{kg}}{\text{m}^3}$	$1.18\frac{\text{kg}}{\text{m}^3}$

Table 6.1: Parameters of the two variations of the cantilever problem.

Kármán vortices and track the vertical displacement of the cantilever tip.

Applying the third-order Corrected Implicit method to Wall's problem, with a timestep of $\Delta t = 0.001$ seconds, we observe the vertical tip displacement shown in figure 6.3, with a maximum tip displacement of 1.13cm. Interpolating linearly between timesteps, we see zero tip displacement at $t = 2.13285$ and $t = 19.99732$, and 57 cycles between these two zeros, giving a frequency of 3.19Hz. Our frequency and maximum displacement agree well with the values obtained in the literature, which uses various other methods and discretizations. In particular, our frequency and amplitude agree almost exactly with those found using the ESDIRK3 method in [30], 3.18Hz and 1.12cm. Similar results are shown for the fifth-order method in figure 6.4, but with a larger timestep. We also apply the same third-order Corrected Implicit method to Hübner's problem, again with a timestep of $\Delta t = 0.001$

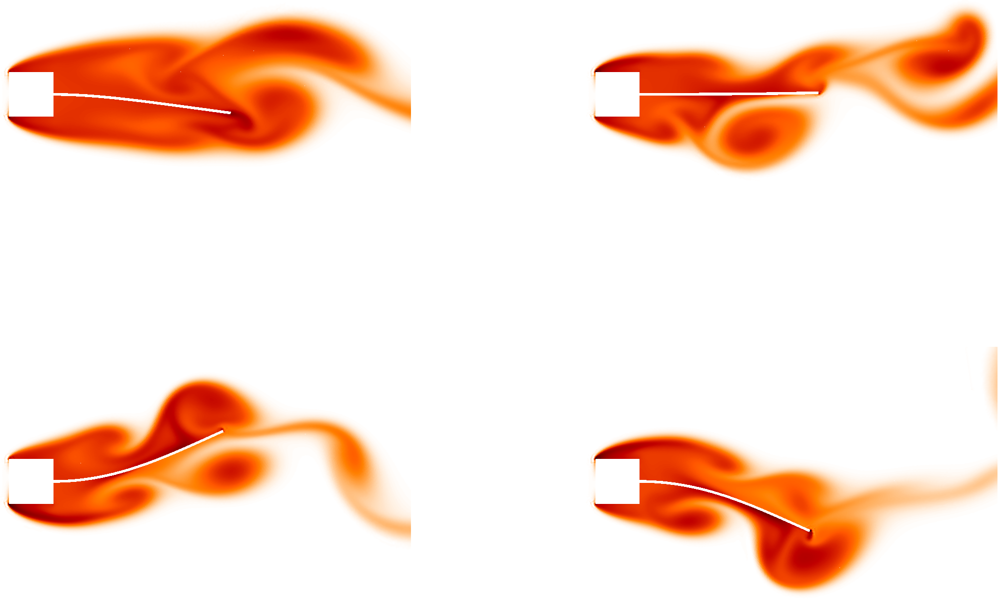


Figure 6.2: Solution to Wall's variation at the beginning of vortex shedding ($t = 0.600s$), in the middle of a flap ($t = 1.820s$), at the highest tip displacement ($t = 2.355s$), and at the lowest dip displacement $t = 2.515s$, with color gradient corresponding to entropy (EDIRK-CIRK5, no G-S iterations, $\Delta t = 0.005$)

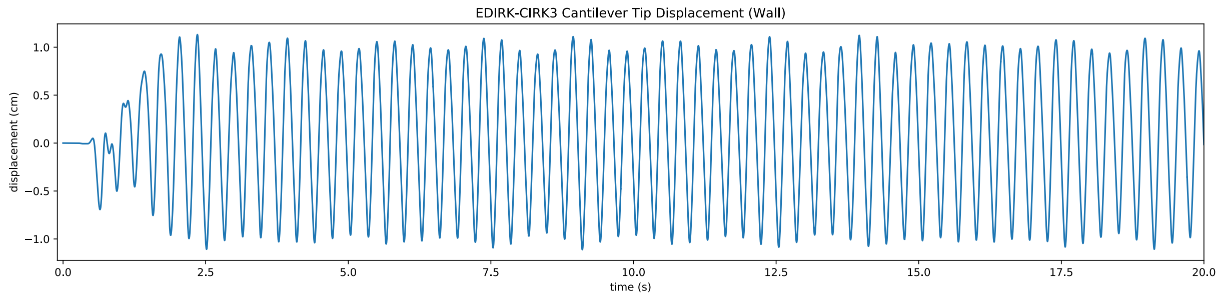


Figure 6.3: Vertical Tip Displacement in centimeters using EDIRK-CIRK3 method with two Gauss-Seidel iterations and timestep $\Delta t = 0.001$: Maximum amplitude is 1.1317cm; frequency 3.1907Hz.

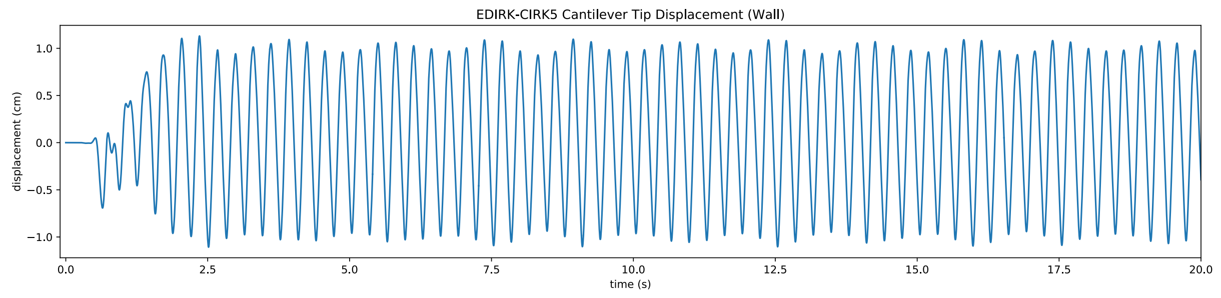


Figure 6.4: Vertical Tip Displacement in centimeters using EDIRK-CIRK5 method with two Gauss-Seidel iterations and timestep $\Delta t = 0.005$: Maximum amplitude is 1.1316cm; frequency 3.1950Hz. (ESDIRK5 with two G-S iterations shows a maximum amplitude of 1.1318cm and a frequency of 3.1871Hz with a timestep of $\Delta t = 0.001$, for comparison.)

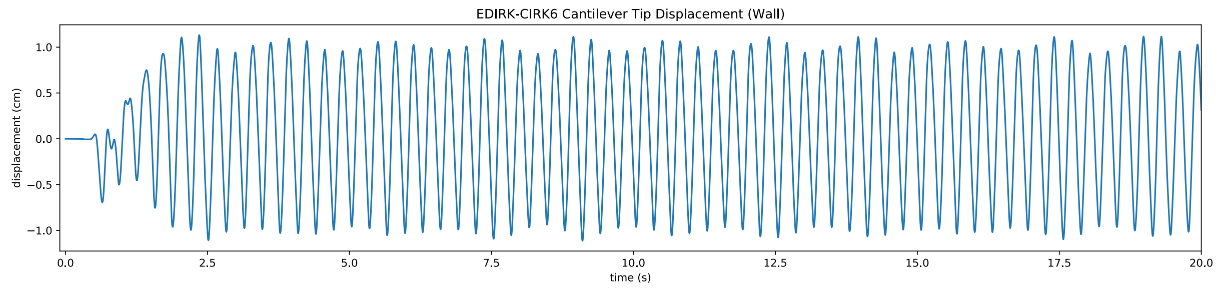


Figure 6.5: Vertical Tip Displacement in centimeters using EDIRK-CIRK6 method with two Gauss-Seidel iterations and timestep $\Delta t = 0.001$: Maximum amplitude is 1.1318cm; frequency 3.1871Hz.

seconds, and show the observed vertical tip displacement shown in figure 6.6. Using the same process, we observe twelve cycles between the zeros at $t = 1.37992$ and $t = 16.96617$, giving a frequency of 0.77Hz. This, along with the maximum tip displacement of 2.23cm, match those found in the initial deflection mode from [38], which observes 2.0cm and 0.8Hz.

Applying the Fast Fourier Transform to the tip displacement values generated using our EDIRK-CIRK method at orders 3, 5, and 6, we observe a second mode of oscillation at 3.417Hz, 3.667Hz, and 3.667Hz respectively, with a much lower amplitude. This agrees with the higher frequency mode found in this system at 3.1Hz in [38], 3.067Hz in [24] and 3.125Hz in [70]. When this analysis is performed on the tip displacement seen using the ESDIRK5 method, shown in figure 6.9, no such secondary frequency is found, suggesting an accuracy advantage with the current method over the comparison ESDIRK method.

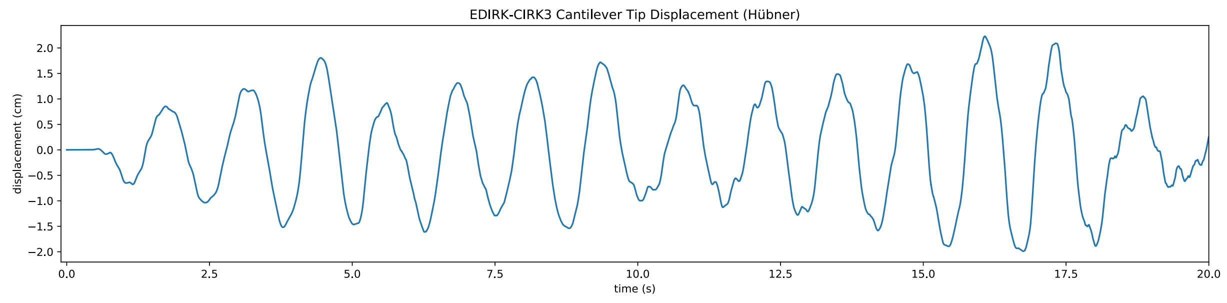


Figure 6.6: Vertical Tip Displacement in centimeters (cm) using EDIRK-CIRK3 method with two Gauss-Seidel iterations and timestep $\Delta t = 0.001$: Maximum amplitude is 2.2296cm; frequency 0.7699Hz.

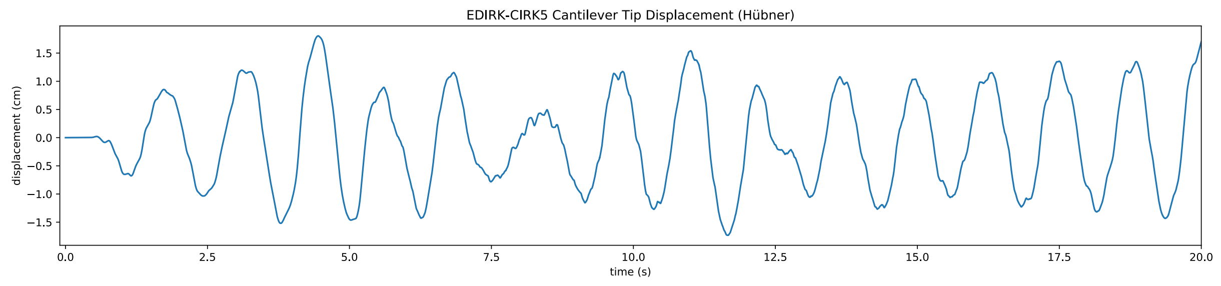


Figure 6.7: Vertical Tip Displacement in centimeters (cm) using ESDIRK5 method with one Gauss-Seidel iteration and timestep $\Delta t = 0.001$: Maximum amplitude is 1.8057cm; frequency 0.7652Hz.

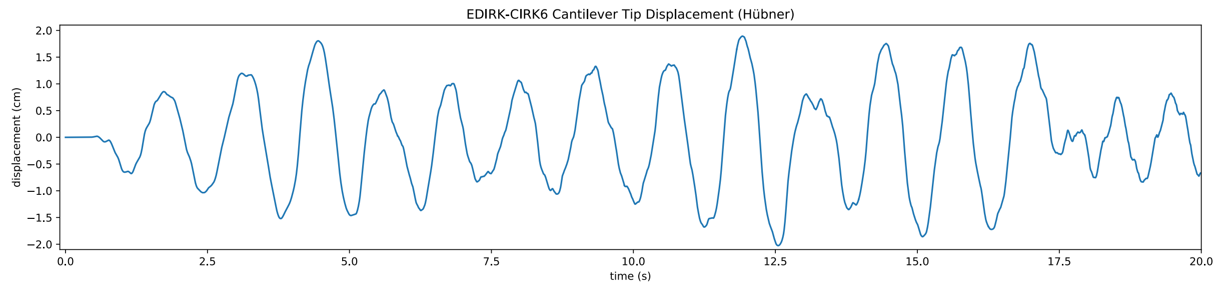


Figure 6.8: Vertical Tip Displacement in centimeters (cm) using EDIRK-CIRK6 method with two Gauss-Seidel iterations and timestep $\Delta t = 0.001$: Maximum amplitude is 2.0263cm; frequency 0.7866Hz.

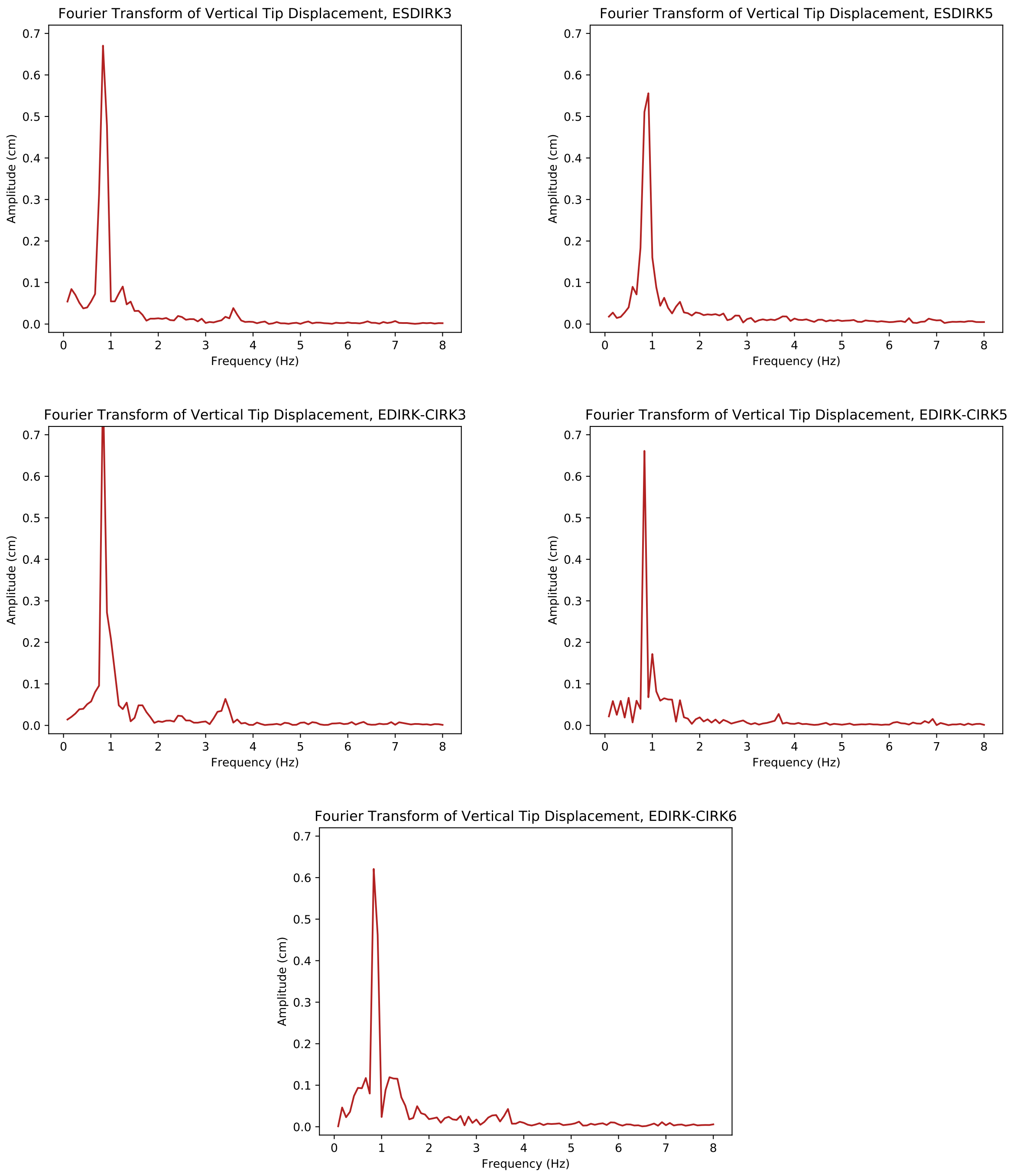


Figure 6.9: Fourier Analysis of Tip Displacement, Hübner's Problem (One additional G-S iteration, $\Delta t = 0.001$, $T = 8.0$ to $T = 20.0$)

In addition to solution frequency, we compare cost and timestep viability of our CIRK method, shown in Table 6.2, using both metric laid out in section 4.2.4, matrix application equivalents and Jacobian assembly equivalents. We find exceptional stability properties in the EDIRK-CIRK5 method, regardless of number Gauss-Seidel iterations, compared to all the other methods studied. Additionally, our EDIRK-CIRK method is significantly cheaper, as compared with the ESDIRK methods using matrix application equivalents, for solving this cantilever problem. This direct comparison is only possible for orders three and five, as no DIRK6 IMEX method is established in [44]. However, we note that though our EDIRK-CIRK6 method is shown in Table 6.2 to be slightly more expensive over the first second of simulation, simulating Hübner's cantilever to $T = 20.0s$ with $\Delta t = 0.001s$ actually costs 9.4% fewer matrix application equivalents and 2.9% fewer matrix equivalent assemblies using EDIRK-CIRK6 than EDIRK-CIRK5, suggesting our sixth-order method would be competitive in cost with any DIRK method of order six.

We find that in the two cantilever problems, where additional Gauss-Seidel iterations are needed, our CIRK method becomes significantly cheaper than the ESDIRK methods from [30], by a factor up to three, when counting matrix application equivalents. Though adding these iterations in both ESDIRK and CIRK methods allows for similar gains in timestep size viability, it is clear from the results found in Table 6.2 that additional Gauss-Seidel iterations increase the cost of the method only a modest 10-20%, whereas the cost of ESDIRK methods must roughly double for the same advantage. (The exception being EDIRK-CIRK5, where G-S iterations are not even necessary.) This gives our EDIRK-CIRK method a considerable cost advantage for solving these problems.

6.2 Tuning Fork

We first apply our method to a two-dimensional tuning fork, whose geometry is shown in figure 6.12. The structure is modelled with Young modulus $E = 200\text{GPa}$, Poisson coefficient $\nu = 0.29$, and density $\rho = 7800\frac{\text{kg}}{\text{m}^3}$. We impose a mesh on the flexible structure with 358 triangular elements. We use the continuous Galerkin framework described in 4.1.2, with a degree $p = 3$ nodal basis. Though it would be possible for this application to get away with a simpler linear elasticity model, we use the neo-Hookean formulation formulation from section 2.4, and reuse the code from the cantilever simulation.

According to the coupling described in section 5.1.1, we interface the tuning fork with a fluid mesh of 1,456 triangular elements, also of degree $p = 3$, upon which we impose the discontinuous Galerkin formulation described in section 4.1.1. A partitioning of this mesh is shown in figure 6.11. Our fluid is modeled as viscous flow by the compressible Navier-Stokes equations, with initial density $1.204\frac{\text{kg}}{\text{m}^3}$, speed of sound $343\frac{\text{m}}{\text{s}}$, a kinematic viscosity of $1.525 \times 10^{-5}\frac{\text{m}^2}{\text{s}}$, and a dynamic viscosity of $1.8306 \times 10^{-5}\frac{\text{kg}}{\text{m}\cdot\text{s}}$. The internal boundary of the fluid domain, where it interfaces with the tuning fork, is modelled with no-slip boundary conditions. The outer boundary of the fluid domain has far-field boundary conditions,

Method	Variation	timestep size (Δt)					
Two G-S Iterations		0.02	0.01	0.005	0.004	0.002	0.001
ESDIRK3	Wall	*	*	*	*	398 (8.6)	450 (14.9)
	Hübner	*	*	341 (3.7)	353 (4.5)	377 (8.0)	451 (14.8)
EDIRK-CIRK3	Wall	*	*	*	*	*	170 (9.3)
	Hübner	*	*	104 (2.3)	108 (2.8)	132 (5.3)	169 (9.4)
ESDIRK5	Wall	*	*	410 (7.7)	422 (9.3)	491 (16.8)	578 (28.2)
	Hübner	*	374 (4.2)	408 (7.7)	419 (9.3)	510 (17.3)	580 (28.2)
EDIRK-CIRK5	Wall	*	*	319 (4.4)	329 (5.3)	391 (9.2)	506 (16.5)
	Hübner	220 (1.6)	265 (2.5)	315 (4.2)	323 (5.1)	386 (9.2)	498 (16.5)
EDIRK-CIRK6	Wall	*	*	*	*	*	497 (16.8)
	Hübner	*	*	175 (4.0)	199 (4.9)	297 (9.1)	487 (17.5)
One G-S Iteration		0.02	0.01	0.005	0.004	0.002	0.001
ESDIRK3	Wall	*	*	*	*	*	*
	Hübner	*	*	*	*	*	*
EDIRK-CIRK3	Wall	*	*	*	*	*	*
	Hübner	*	*	*	*	112 (4.3)	139 (7.4)
ESDIRK5	Wall	*	*	*	*	*	*
	Hübner	*	*	*	*	254 (8.6)	290 (14.1)
EDIRK-CIRK5	Wall	*	*	284 (3.4)	291 (4.1)	338 (7.2)	432 (13.4)
	Hübner	197 (1.3)	241 (2.0)	287 (3.4)	293 (4.1)	345 (7.5)	436 (13.4)
EDIRK-CIRK6	Wall	*	*	*	*	*	*
	Hübner	*	*	*	*	*	*

Table 6.2: Cumulative Fluid Solve Cost in thousands of Matrix Application Equivalents (Jacobian matrix equivalent assemblies in parentheses) to $T = 1.0$, Cantilever Problem Note that * marks method failure. Gauss-Seidel iterations are described in algorithm 4 and [30].

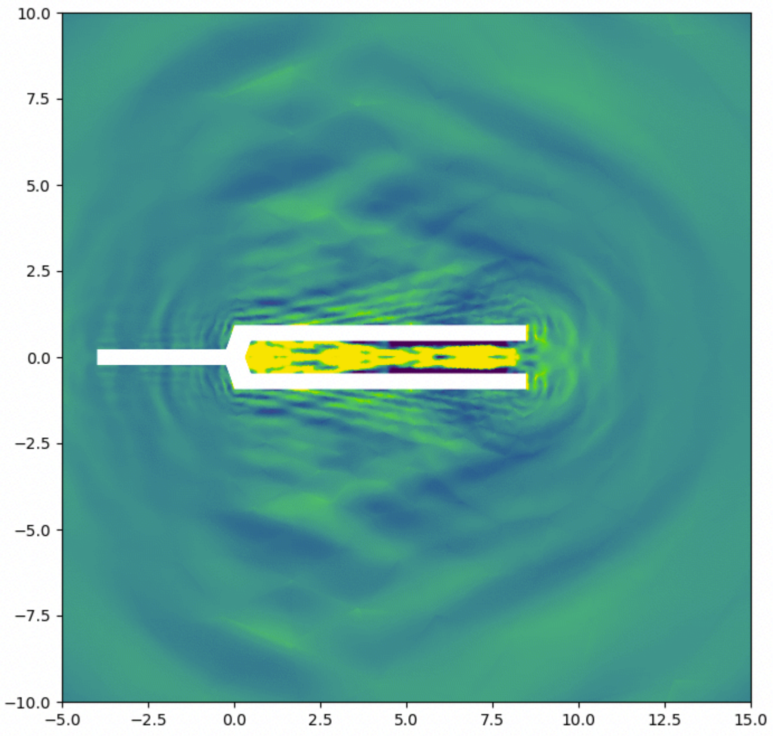


Figure 6.10: Fork Solution at time $t = 0.0005$ seconds. The scheme used is EDIRK-CIRK5 with timestep $\Delta t = 0.001$ milliseconds. The color denotes pressure.

described in section 2.1.2 with fluid velocity zero. The far-field boundary conditions we use are only able to absorb planar waves which approach the boundary orthogonally. Though so-called *absorbing boundary conditions* would prohibit the reflection of pressure waved created by the tuning fork back into the computational domain, our boundary conditions prove a sufficient approximation in practice.

We integrate the system forward in time using both the reference ESDIRK from [30], as well as our CIRK method. For the CIRK method, no attempt was made at explicit prediction, as the system is considered too stiff. For both methods, we use the Newton-Krylov methods described in section 4.2 with an error tolerance of 10^{-8} for Newton's method and a relative tolerance of 10^{-5} for the GMRES iterations. The initial conditions consist of the fluid at rest, and an initial perturbation of the velocity of the structure. A solution is shown in figure 6.10.

From the tuning fork, we wish to extract an audible pitch. To do this, we plot the pressure

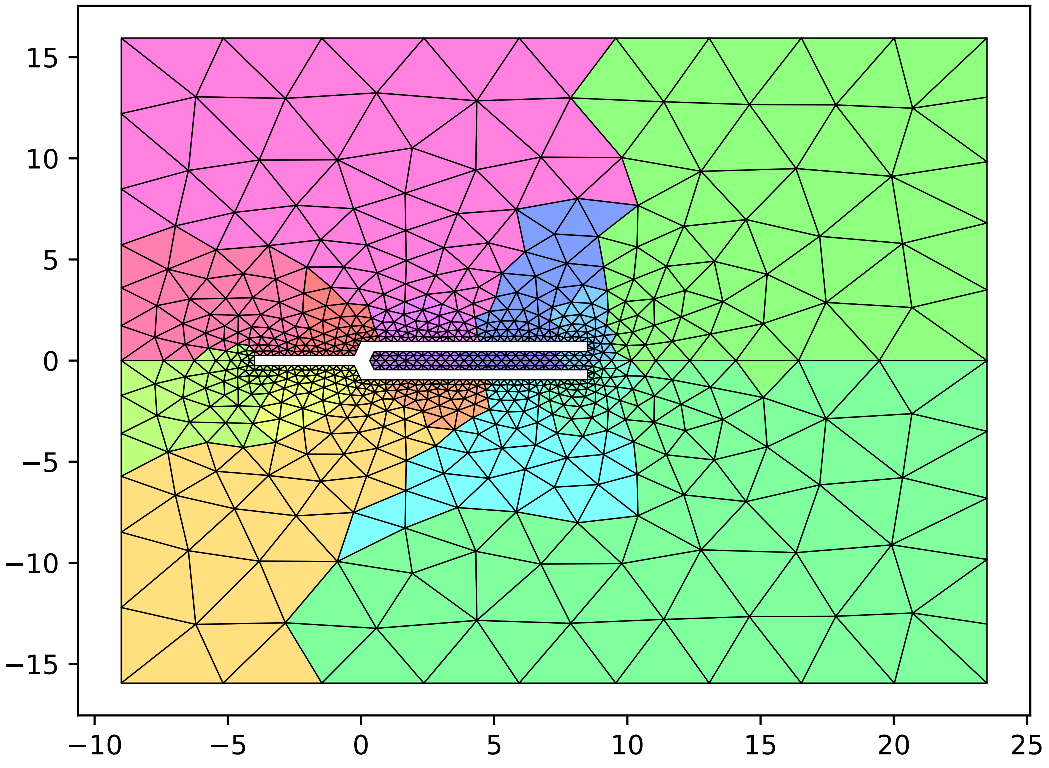


Figure 6.11: Tuning Fork fluid mesh with sixteen partitions

at one of the nodes near the base of the tuning fork over time, obtained from our EDIRK-CIRK5 method with timestep $\Delta t = 0.01$ milliseconds. This is shown in figure 6.13, which demonstrates a clear decay of amplitude in the sound wave. We perform a Fourier analysis of the second half of this simulation, from 50-100 milliseconds, and see a clear fundamental peak at 600Hz. We present this Fourier analysis in figure 6.14, both in linear scale from 20Hz to 20,000Hz, and in log scale from 200Hz to 20,000Hz. (20Hz-20,000Hz is generally considered to be the full range of human hearing.) In addition to the fundamental peak at 600Hz, we see a secondary peak at 3640Hz, and a tertiary peak at 10200Hz. This represents a note between D_5 and $D^\sharp_5$, with two overtones, one two octaves and a fifth higher, and the other three octaves and a ninth higher, as shown on the right side of figure 6.14.

For comparison, we run the same simulation using the ESDIRK5 method from [30]. As the timestep $\Delta t = 0.01$ milliseconds is not viable for this method, we use $\Delta t = 0.005$. We track the pressure in the same location, and perform a Fourier analysis on the portion of

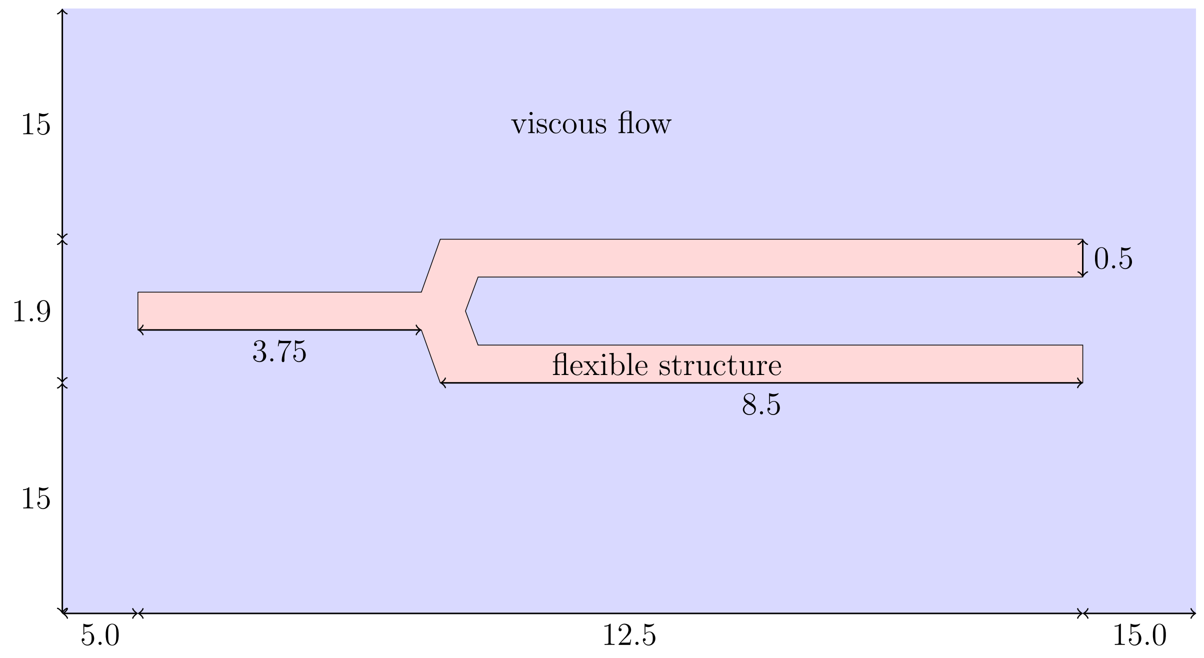
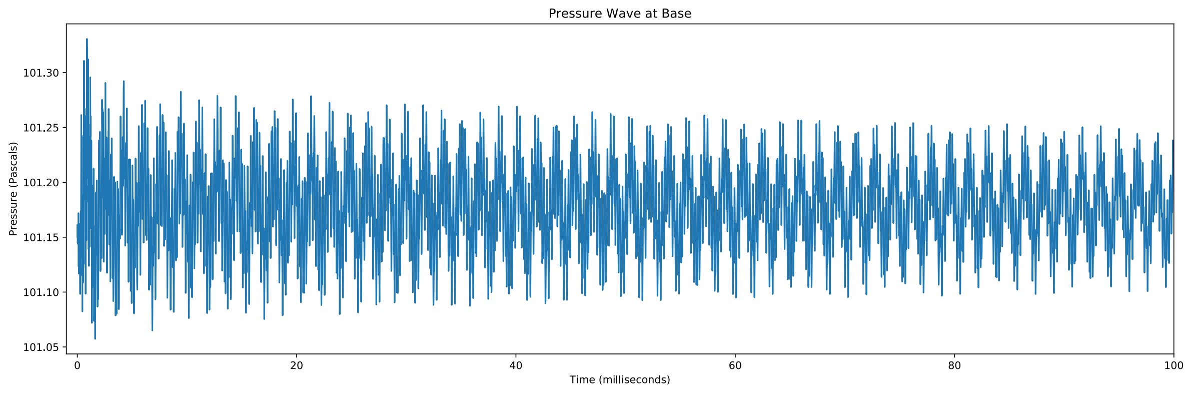


Figure 6.12: Two-dimensional Cantilever System, with fixed left edge (dimensions in cm, fluid boundary not to scale)

the wave from 50-100 milliseconds. The results are shown in figure 6.14. The frequencies observed in the ESDIRK5 simulation correspond almost exactly with those from the EDIRK-CIRK5 method, with the peaks at the same frequencies, 600Hz, 3640Hz, and 10200Hz. The difference comes only in the amplitude of the vibrations, where the ESDIRK5 method shows greater variation in pressure for the first few milliseconds, followed by steeper decay.

Unfortunately, though finite element analysis of tuning forks has been done in three dimensions in [41] and [31], there is no existing literature for this particular test problem, so measurements of accuracy cannot be compared to previous work. However, using the metrics described in section 4.2.4, we are able to compare the computational work of our EDIRK-CIRK method with that of the existing ESDIRK method. Using fifth-order schemes, with $\Delta t = 10^{-3}$ milliseconds, our EDIRK-CIRK5 method requires 1,599,353 matrix applications equivalents and 120,152 Jacobian assembly equivalents, whereas the ESDIRK5 method requires 1,109,757 matrix application equivalents and 140,305 Jacobian assembly equivalents. This means the EDIRK-CIRK5 method requires 44.1% more matrix application equivalents, but 14.4% fewer Jacobian assemblies. Assuming that a computer architecture results in Jacobian assembly being thirty times more expensive than a matrix application equivalent, our EDIRK-CIRK5 method is less expensive in this simulation. The pattern of our EDIRK-CIRK5 method requiring more matrix application equivalents but fewer Jacobian assembly equivalents than the ESDIRK5 method also holds for larger timesteps. Furthermore, our



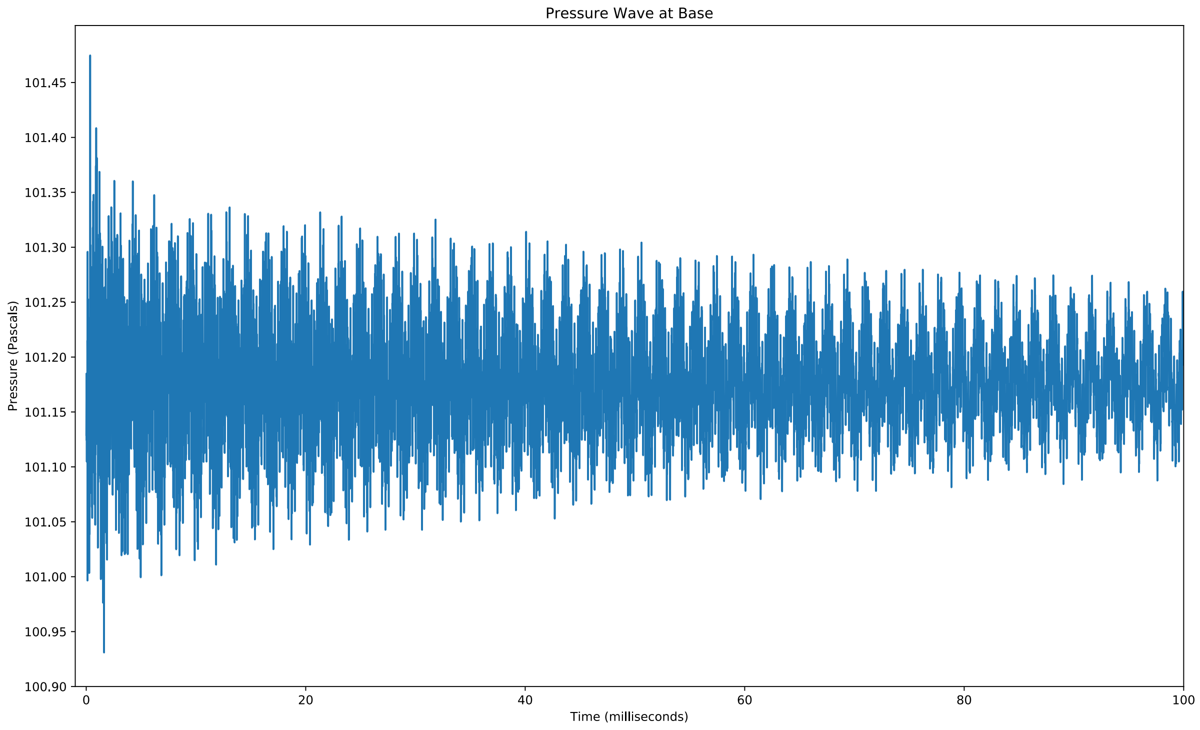


Figure 6.13: Pressure during first hundred milliseconds of Tuning Fork Simulation
Top: EDIRK-CIRK5 Metho with $\Delta t = 0.01$ milliseconds
Bottom: ESDIRK5 with $\Delta t = 0.005$ milliseconds
Both simulations use two Gauss-Seidel subiterations. Though the ESDIRK method exhibits a larger initial amplitude, the two methods show similar wave patterns and sound decay.

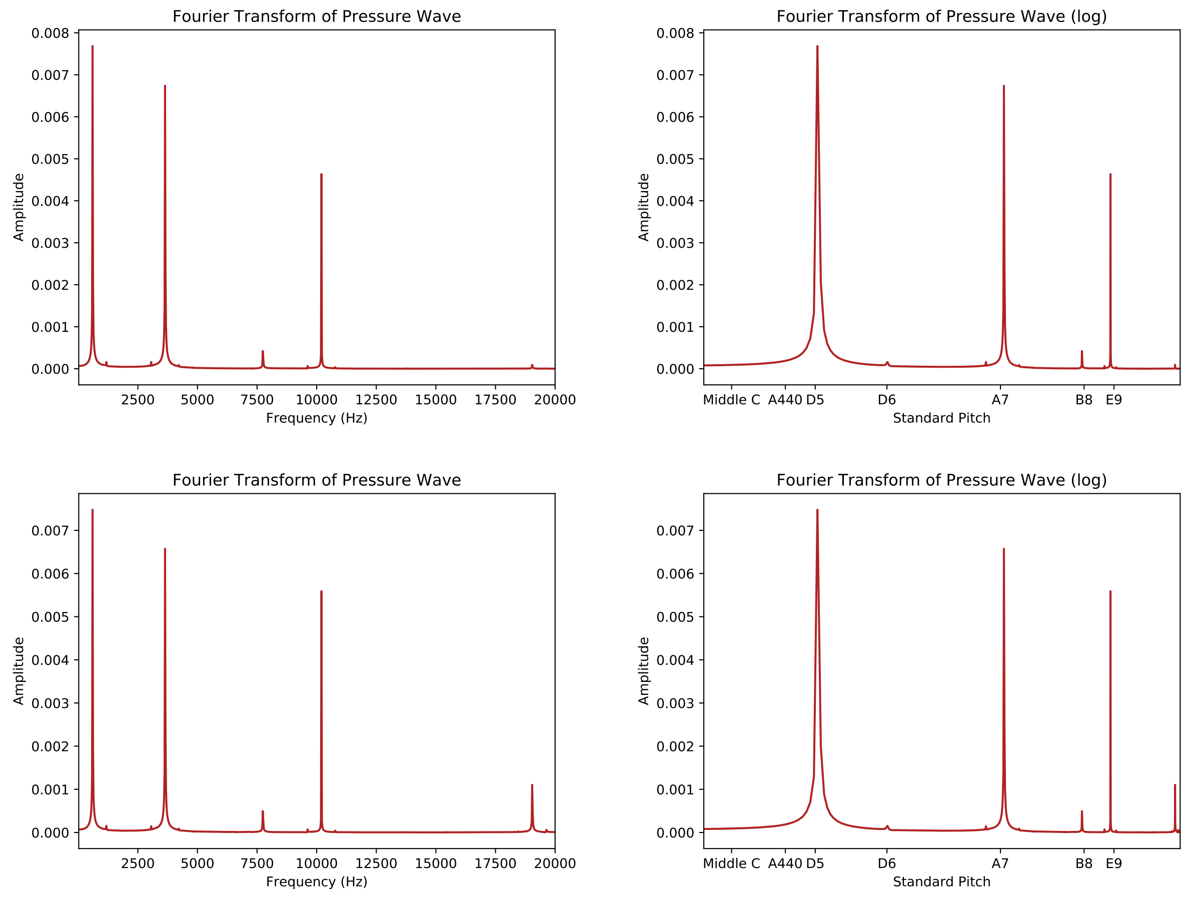


Figure 6.14: Fourier Transform of Tuning Fork Pressure Waves from Figure 6.13
The transform of the wave generated using the EDIRK-CIRK5 method is shown above, while the transform of the ESDIRK5-generated wave is shown below. The frequency axis is scaled linearly on the left and in log scale on the right, with stanard A440 musical pitch labelled. The frequency peaks of the two methods match exactly in position, differing only in size.

Method	timestep size, milliseconds (Δt)					
1 G-S Iteration	0.1	0.08	0.04	0.02	0.01	0.005
ESDIRK5	*	*	*	*	*	403 (28.3)
EDIRK-CIRK5	239 (1.6)	241 (1.9)	234 (3.3)	250 (6.1)	335 (12.1)	508 (24.5)
2 G-S Iterations	0.1	0.08	0.04	0.02	0.01	0.005
ESDIRK5	*	*	*	*	*	806 (56.6)
EDIRK-CIRK5	258 (1.9)	256 (2.2)	259 (4.0)	290 (7.6)	409 (15.1)	634 (30.5)
3 G-S Iterations	0.1	0.08	0.04	0.02	0.01	0.005
ESDIRK5	*	*	*	*	*	1209 (85.0)
EDIRK-CIRK5	276 (2.2)	277 (2.6)	287 (4.6)	291 (7.6)	412 (15.3)	634 (30.5)

Table 6.3: Turning Fork Problem: Cumulative Fluid Solve Cost in thousands of Matrix Application Equivalents (Jacobian matrix equivalent assemblies in parentheses) to $T = 0.01s$ Note that * marks method failure. None of the methods studies were viable at $\Delta t = 0.2$ milliseconds. Gauss-Seidel subiterations are described in algorithm 4 and [30].

fifth-order method also exhibits a significant advantage in stability, as shown in table 6.3, allowing for timesteps that are twenty times greater.

One proxy for the accuracy of the EDIRK-CIRK5 method, as compared to solving the monolithic system, is the number of additional Newton iterations required when adding Gauss-Seidel subiterations. If the CIRK method solves the monolithic system to within the Newton iteration tolerance, then our implementation will recognise this and initiate no additional Newton iterations at the fluid solve stage of the next subiteration. We see in table 6.3 that the first additional Gauss-Seidel subiteration adds somewhere between twenty and twenty-five percent more Newton iterations for the CIRK5 method, suggesting that there is some discrepancy between the solution calculated with two Gauss-Seidel subiterations and that which would be calculated by applying the fifth-order Radau IIA method directly to the monolithic system. With three Gauss-Seidel subiterations, however, we see almost no additional computational cost at the smaller timesteps, culminating in only nine extra Newton iterations over 20,000 steps for the smallest timestep of $\Delta t = 0.005$ milliseconds. This suggests that in this application, our method, with two subiterations, is no worse than a monolithic method would be, and that adding a third Gauss-Seidel subiteration is generally useless.

As section 5.4 shows clearly, our method achieves its design-order, without the use of Gauss-Seidel subiterations. Though they cannot increase the order of accuracy of the method, adding one can in some instances be used to increase the stability of the method, in order to use larger timesteps. We have not observed this with the fifth-order method, but have in applications of third and sixth order.

Chapter 7

Conclusions & Future Work

We have presented a high order accurate fully implicit Runge-Kutta method for time integration of fluid-structure interaction problems. This method can be paired with two different prediction methods, tackling a range of different problems. As it is a partitioned method, as opposed to monolithic, it allows for the use of standard fluid and structure solvers, without the need for specialized solvers.

We apply our method to a variety of problems of interest, including the standard cantilever problem. The results agree with existing literature, without more computational cost than the comparison diagonally implicit Runge-Kutta method implemented. In fact, when subiterations are necessary for stability, these come at a significantly lower cost than those of the comparison method, presenting a significant advantage, while maintaining the advantages of a fully implicit Runge-Kutta method.

Moving forward, it would be advantageous to apply this method to three-dimensional problems. Additionally, investigation of mesh-motion strategies other than radial basis functions could allow for the application of this method to problems with larger structure deformations. Our method could also be applied with a stage-parallel mesh to decrease the computational cost of the IRK solves.

This method could also be modified in several ways. Though we have implemented it with only two different prediction methods, any number of novel prediction methods could be used, allowing even more efficient time integration of fluid-structure interaction problems. Finally, the Implicit-Explicit Prediction methods we develop in chapter 3 could be used for the time integration of multi-physics problems using IRK methods, in a predictor-corrector framework similar to our CIRK method.

Bibliography

[1] Othmar H Amman, Theodore von Kármán, and Glenn B Woodruff. "The failure of the Tacoma Narrows bridge". In: (1941).

[2] Arun Arjunan et al. "Development of a 3D finite element acoustic model to predict the sound reduction index of stud based double-leaf walls". In: *Journal of Sound and Vibration* 333.23 (2014), pp. 6140–6155.

[3] Douglas N Arnold et al. "Unified analysis of discontinuous Galerkin methods for elliptic problems". In: *SIAM journal on numerical analysis* 39.5 (2002), pp. 1749–1779.

[4] Uri M Ascher, Steven J Ruuth, and Raymond J Spiteri. "Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations". In: *Applied Numerical Mathematics* 25.2-3 (1997), pp. 151–167.

[5] Joseph W Bahlman, Sharon M Swartz, and Kenneth S Breuer. "Design and characterization of a multi-articulated robotic bat wing". In: *Bioinspiration & biomimetics* 8.1 (2013), p. 016009.

[6] Monika Balázsová et al. "Space-time discontinuous Galerkin method for the solution of fluid-structure interaction". In: *Appl. Math.* 63.6 (2018), pp. 739–764. ISSN: 0862-7940. DOI: 10.21136/AM.2018.0139-18.

[7] Jeffrey W Banks, William D Henshaw, and Donald W Schwendeman. "An analysis of a new stable partitioned algorithm for FSI problems. Part I: Incompressible flow and elastic solids". In: *Journal of Computational Physics* 269 (2014), pp. 108–137.

[8] Jeffrey W Banks, William D Henshaw, and Donald W Schwendeman. "An analysis of a new stable partitioned algorithm for FSI problems. Part II: Incompressible flow and structural shells". In: *Journal of Computational Physics* 268 (2014), pp. 399–416.

[9] Yuri Bazilevs et al. "Fluid–structure interaction modeling for fatigue-damage prediction in full-scale wind-turbine blades". In: *Journal of Applied Mechanics* 83.6 (2016).

[10] Armin Beckert and Holger Wendland. "Multivariate interpolation for fluid-structure-interaction problems using radial basis functions". In: *Aerospace Science and Technology* 5.2 (2001), pp. 125–134.

[11] Philipp Birken et al. "A time-adaptive fluid-structure interaction method for thermal coupling". In: *Comput. Vis. Sci.* 13.7 (2010), pp. 331–340. ISSN: 1432-9360. DOI: 10.1007/s00791-010-0150-4.

[12] Aukje de Boer, Martijn S. van der Schoot, and Hester Bijl. "Mesh deformation based on radial basis function interpolation". In: *Computers & structures* 85.11-14 (2007), pp. 784–795.

[13] Joris Bols et al. "A computational method to assess the in vivo stresses and unloaded configuration of patient-specific blood vessels". In: *Journal of computational and Applied mathematics* 246 (2013), pp. 10–17.

[14] Martina Bukač and Sunčica Čanić. "A partitioned numerical scheme for fluid–structure interaction with slip". In: *Mathematical Modelling of Natural Phenomena* 16 (2021), p. 8.

[15] Martina Bukač, Sunćica Čanić, and Boris Muha. "A partitioned scheme for fluid–composite structure interaction problems". In: *Journal of Computational Physics* 281 (2015), pp. 493–517.

[16] Martina Bukač and Catalin Trenchea. "Adaptive, Second-Order, Unconditionally Stable Partitioned Method for Fluid-Structure Interaction". In: *misc.* (2021). URL: https://www.mathematics.pitt.edu/sites/default/files/2021_variableDtFSIthin_technicalreport.pdf.

[17] John C Butcher. "Implicit runge-kutta processes". In: *Mathematics of Computation* 18.85 (1964), pp. 50–64.

[18] John Charles Butcher. "A history of Runge-Kutta methods". In: *Applied numerical mathematics* 20.3 (1996), pp. 247–260.

[19] John Charles Butcher. "Trees and B-series". In: *Numerical Algorithms* 81.4 (2019), pp. 1311–1325.

[20] Huade Cao, Magdi Mohareb, and Ioan Nistor. "Partitioned water hammer modeling using the block Gauss–Seidel algorithm". In: *Journal of Fluids and Structures* 103 (2021), p. 103260.

[21] Jeff R. Cash and Alan H. Karp. "A Variable Order Runge-Kutta Method for Initial Value Problems with Rapidly Varying Right-Hand Sides". In: *ACM Trans. Math. Softw.* 16.3 (Sept. 1990), pp. 201–222. ISSN: 0098-3500. DOI: 10.1145/79505.79507.

[22] Arthur Cayley. *On the theory of the analytical forms called trees, Math.* 1897.

[23] Alexandre J Chorin et al. "Product formulas and numerical algorithms". In: *Communications on Pure and Applied Mathematics* 31.2 (1978), pp. 205–256.

[24] Jean-François Cori et al. "High-order implicit Runge-Kutta time integrators for fluid-structure interactions". In: *International Journal for Numerical Methods in Fluids* 78.7 (2015), pp. 385–412. DOI: 10.1002/fld.4020.

[25] Alvaro L. De Bortoli. "Aeroelastic analysis of panels in compressible flows". In: *Journal of Fluids and Structures* 20.2 (2005), pp. 189–195. ISSN: 0889-9746. DOI: `h10.1016/j.jfluidstructs.2004.10.011`.

[26] Wulf G. Dettmer and Djordje Perić. "A computational framework for fluid-structure interaction: Finite element formulation and applications". In: *Computer Methods in Applied Mechanics and Engineering* 195.41 (2006). John H. Argyris Memorial Issue. Part II, pp. 5754–5779. ISSN: 0045-7825. DOI: `10.1016/j.cma.2005.10.019`.

[27] Wulf G. Dettmer and Djordje Perić. "A new staggered scheme for fluid-structure interaction". In: *Internat. J. Numer. Methods Engrg.* 93.1 (2013), pp. 1–22. ISSN: 0029-5981. DOI: `10.1002/nme.4370`.

[28] Charbel Farhat and Michel Lesoinne. "Two efficient staggered algorithms for the serial and parallel solution of three-dimensional nonlinear transient aeroelastic problems". In: *Computer Methods in Applied Mechanics and Engineering* 182.3 (2000), pp. 499–515. ISSN: 0045-7825. DOI: `10.1016/S0045-7825(99)00206-6`.

[29] Carlos A. Felippa, Kwang-Chun Park, and Charbel Farhat. "Partitioned analysis of coupled mechanical systems". In: *Computer Methods in Applied Mechanics and Engineering* 190.24 (2001). Advances in Computational Methods for Fluid-Structure Interaction, pp. 3247–3270. ISSN: 0045-7825. DOI: `10.1016/S0045-7825(00)00391-1`.

[30] Bradley Froehle and Per-Olof Persson. "A high-order discontinuous Galerkin method for fluid-structure interaction with efficient implicit-explicit time stepping". In: *J. Comput. Phys.* 272 (2014), pp. 455–470. ISSN: 0021-9991. DOI: `10.1016/j.jcp.2014.03.034`.

[31] Bradley Michael Froehle. "High-Order Discontinuous Galerkin Fluid-Structure Interaction Methods". PhD thesis. University of California, Berkeley, 2013.

[32] Philippe Geuzaine, Céline Grandmont, and Charbel Farhat. "Design and analysis of ALE schemes with provable second-order time-accuracy for inviscid and viscous flow simulations". In: *Journal of Computational Physics* 191.1 (2003), pp. 206–227. ISSN: 0021-9991. DOI: `10.1016/S0021-9991(03)00311-5`.

[33] Ernst Hairer and Gerhard Wanner. "On the Butcher group and general multi-value methods". In: *Computing* 13.1 (1974), pp. 1–15.

[34] Peter Hansbo, Joakim Hermansson, and Thomas Svedberg. "Nitsche's method combined with space-time finite elements for ALE fluid-structure interaction problems". In: *Comput. Methods Appl. Mech. Engrg.* 193.39-41 (2004), pp. 4195–4206. ISSN: 0045-7825. DOI: `10.1016/j.cma.2003.09.029`.

[35] Matthias Heil. "An efficient solver for the fully coupled solution of large-displacement fluid–structure interaction problems". In: *Computer Methods in Applied Mechanics and Engineering* 193.1-2 (2004), pp. 1–23.

[36] Guillaume Houzeaux et al. “Parallel Multiphysics Coupling: Algorithmic and Computational Performances”. In: *International Journal of Computational Fluid Dynamics* 34.7-8 (2020), pp. 486–507.

[37] Daniel Z. Huang, Per-Olof Persson, and Matthew J. Zahr. “High-order, linearly stable, partitioned solvers for general multiphysics problems based on implicit-explicit Runge-Kutta schemes”. In: *Comput. Methods Appl. Mech. Engrg.* 346 (2019), pp. 674–706. ISSN: 0045-7825. DOI: 10.1016/j.cma.2018.09.015.

[38] Björn Hübner, Elmar Walhorn, and Dieter Dinkler. “A monolithic approach to fluid-structure interaction using space-time finite elements”. In: *Computer Methods in Applied Mechanics and Engineering* 193.23 (2004), pp. 2087–2104. ISSN: 0045-7825. DOI: 10.1016/j.cma.2004.01.024.

[39] Linda Ingebrigtsen et al. “On the order and accuracy of operator splitting for a fluid-structure interaction problem”. In: *International Journal of Nonlinear Sciences and Numerical Simulation* 4.3 (2003), pp. 209–218.

[40] Yih-Jena Jan and Tony Sheu. “Finite element analysis of vortex sheding oscillations from cylinders in the straight channel”. In: *Computational Mechanics* 33 (Dec. 2004), pp. 81–94. DOI: 10.1007/s00466-003-0502-8.

[41] Soon-Suck Jarng and You-Jung Kwon. “TUNING Fork Analysis and Design by FEM AND BEM”. In: *Proceedings of the Korean Society for Noise and Vibration Engineering Conference.* The Korean Society for Noise and Vibration Engineering. 2003, pp. 1201–1204.

[42] MM Joosten, Wulf G. Dettmer, and Djordje Perić. “Analysis of the block Gauss–Seidel solution procedure for a strongly coupled model problem with reference to fluid–structure interaction”. In: *International Journal for Numerical Methods in Engineering* 78.7 (2009), pp. 757–778.

[43] George Karypis and Vipin Kumar. “A fast and high quality multilevel scheme for partitioning irregular graphs”. In: *SIAM Journal on scientific Computing* 20.1 (1998), pp. 359–392.

[44] Christopher A. Kennedy and Mark H. Carpenter. “Additive Runge-Kutta schemes for convection-diffusion-reaction equations”. In: *Applied Numerical Mathematics* 44.1 (2003), pp. 139–181. ISSN: 0168-9274. DOI: 10.1016/S0168-9274(02)00138-1.

[45] Artem Korobenko et al. “Structural mechanics modeling and FSI simulation of wind turbines”. In: *Mathematical Models and Methods in Applied Sciences* 23.02 (2013), pp. 249–272.

[46] Martin Wilhelm Kutta. *Beitrag zur näherungsweisen Integration totaler Differentialgleichungen.* Teubner, 1901.

[47] Andrea La Spina et al. “A weakly compressible hybridizable discontinuous Galerkin formulation for fluid–structure interaction problems”. In: *Computer Methods in Applied Mechanics and Engineering* 372 (2020), p. 113392.

[48] Wendi Liu et al. "A Parallel Partitioned Approach on Fluid-Structure Interaction Simulation Using the Multiscale Universal Interface Coupling Library". In: *14th WCCM-ECCOMAS Congress 2020*. Vol. 1400. 2021.

[49] Maria Lukáčová-Medvid'ová, Gabriela Rusnáková, and Anna Hundertmark-Zaušková. "Kinematic splitting algorithm for fluid-structure interaction in hemodynamics". In: *Computer Methods in Applied Mechanics and Engineering* 265 (2013), pp. 83–106. ISSN: 0045-7825. DOI: `10.1016/j.cma.2013.05.025`.

[50] Joaquim RRA Martins, Juan J Alonso, and James J Reuther. "High-fidelity aerostructural design optimization of a supersonic business jet". In: *Journal of Aircraft* 41.3 (2004), pp. 523–530.

[51] Hermann G. Matthies and Jan Steindorf. "Partitioned strong coupling algorithms for fluid-structure interaction". In: *Computers & Structures* 81.8 (2003). K.J Bathe 60th Anniversary Issue, pp. 805–812. ISSN: 0045-7949. DOI: `10.1016/S0045-7949(02)00409-1`.

[52] Matthias Mayr et al. "A temporal consistent monolithic approach to fluid-structure interaction enabling single field predictors". In: *SIAM Journal on Scientific Computing* 37.1 (2015), B30–B59.

[53] Jianguo Ning et al. "A novel fluid-structure interaction algorithm for compressible flows and deformable structures". In: *Journal of Computational Physics* 426 (2021), p. 109921.

[54] Vivek Ojha, Krzysztof J Fidkowski, and Carlos ES Cesnik. "Adaptive High-Order Fluid-Structure Interaction Simulations with Reduced Mesh-Motion Errors". In: *AIAA Journal* (2021), pp. 1–19.

[55] Oyekola Oyekole, Catalin Trenchea, and Martina Bukac. "A second-order in time approximation of fluid-structure interaction problem". In: *SIAM Journal on Numerical Analysis* 56.1 (2018), pp. 590–613.

[56] Will Pazner and Per-Olof Persson. "Stage-parallel fully implicit Runge-Kutta solvers for discontinuous Galerkin fluid simulations". In: *J. Comput. Phys.* 335 (2017), pp. 700–717. ISSN: 0021-9991. DOI: `10.1016/j.jcp.2017.01.050`.

[57] Jaime Peraire and Per-Olof Persson. "High-order discontinuous Galerkin methods for CFD". In: *Adaptive high-order methods in computational fluid dynamics*. World Scientific, 2011, pp. 119–152.

[58] Jaime Peraire and Per-Olof Persson. "The compact discontinuous Galerkin (CDG) method for elliptic problems". In: *SIAM Journal on Scientific Computing* 30.4 (2008), pp. 1806–1824.

[59] Per-Olof Persson. "Scalable parallel Newton-Krylov solvers for discontinuous Galerkin discretizations". In: *47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*. 2009, p. 606.

[60] Per-Olof Persson, Javier Bonet, and Jaime Peraire. “Discontinuous Galerkin solution of the Navier–Stokes equations on deformable domains”. In: *Computer Methods in Applied Mechanics and Engineering* 198.17-20 (2009), pp. 1585–1595.

[61] Serge Piperno, Charbel Farhat, and Bernard Larrouturou. “Partitioned procedures for the transient solution of coupled aeroelastic problems. I. Model problem, theory and two-dimensional application”. In: *Comput. Methods Appl. Mech. Engrg.* 124.1-2 (1995), pp. 79–112. ISSN: 0045-7825. DOI: `10.1016/0045-7825(95)92707-9`.

[62] Ekkehard Ramm and Wolfgang Wall. “Fluid-structure interaction based upon a stabilized (ALE) finite element method”. In: *4th World Congress on Computational Mechanics: New Trends and Applications, CIMNE, Barcelona.* Ed. by E. N. Dvorkin S. R. Idelsohn E. Oñate. 1998, pp. 1–20.

[63] Philip L Roe. “Approximate Riemann solvers, parameter vectors, and difference schemes”. In: *Journal of computational physics* 43.2 (1981), pp. 357–372.

[64] Carl Runge. “Über die numerische Auflösung von Differentialgleichungen”. In: *Mathematische Annalen* 46.2 (1895), pp. 167–178.

[65] Sebastian Schöps. “Iterative Schemes for Coupled Multiphysical Problems in Electrical Engineering”. In: *IFAC-PapersOnLine* 48.1 (2015), pp. 165–167.

[66] Wen-Bin Shangguan and Zhen-Hua Lu. “Modelling of a hydraulic engine mount with fluid–structure interaction finite element analysis”. In: *Journal of sound and vibration* 275.1-2 (2004), pp. 193–221.

[67] Kenji Takizawa and Tayfun E. Tezduyar. “Space-time fluid-structure interaction methods”. In: *Math. Models Methods Appl. Sci.* 22.suppl. 2 (2012), pp. 1230001, 49. ISSN: 0218-2025. DOI: `10.1142/S0218202512300013`.

[68] Paul Dennis Thomas and C. K. Lombard. “Geometric conservation law and its application to flow computations on moving grids”. In: *AIAA journal* 17.10 (1979), pp. 1030–1037.

[69] Silvio Tschisgale and Jochen Fröhlich. “An immersed boundary method for the fluid-structure interaction of slender flexible structures in viscous fluid”. In: *Journal of Computational Physics* 423 (2020), p. 109801.

[70] Jesús Gerardo Valdéz Vázquez, Eugenio Oñate Ibáñz de Navarro, and Juan Miquel Canet. “Nonlinear Analysis of Orthotropic Membrane and Shell Structures Including Fluid-Structure Interaction”. PhD thesis. Universitat Politècnica de Catalunya, 2007. URL: `https://upcommons.upc.edu/bitstream/handle/2117/94177/01JGVV01de01.pdf`.

[71] Alexander H. van Zuijlen and Hester Bijl. “Implicit and explicit higher order time integration schemes for structural dynamics and fluid-structure interaction computations”. In: *Computers & Structures* 83.2 (2005). Advances in Analysis of Fluid Structure Interaction, pp. 93–105. ISSN: 0045-7949. DOI: `10.1016/j.compstruc.2004.06.003`.

[72] Alexander H. van Zuijlen, S. Bosscher, and Hester Bijl. "Two level algorithms for partitioned fluid-structure interaction computations". In: *Computer Methods in Applied Mechanics and Engineering* 196.8 (2007). Domain Decomposition Methods: recent advances and new challenges in engineering, pp. 1458–1470. ISSN: 0045-7825. DOI: 10.1016/j.cma.2006.03.014.

[73] Michel Visonneau et al. "Interaction Fluide-Structure pour les corps élancés". In: *Congrès français de mécanique.* AFM, Maison de la Mécanique, 39/41 rue Louis Blanc-92400 Courbevoie. 2009.

[74] Wolfgang Wall. "Fluid-Struktur-Interaktion mit Stabilisierten Finiten Elementen". PhD thesis. Universität Stuttgart Institut für Baustatik, Jan. 2000. DOI: 10.18419/opus-127.

[75] Clare A. C. Wood et al. "A partitioned coupling approach for dynamic fluid-structure interaction with applications to biological membranes". In: *International Journal for Numerical Methods in Fluids* 57.5 (2008), pp. 555–581. DOI: 10.1002/fld.1815.

[76] Clare A. C. Wood et al. "Partitioned block-Gauss-Seidel coupling for dynamic fluid-structure interaction". In: *Computers & Structures* 88.23 (2010). Special Issue: Association of Computational Mechanics - United Kingdom, pp. 1367–1382. ISSN: 0045-7949. DOI: 10.1016/j.compstruc.2008.08.005.

[77] Chunfeng Zhao et al. "Sensitive analysis of water levels and air intakes on natural frequency of AP1000 nuclear island building considering FSI effects". In: *Annals of Nuclear Energy* 78 (2015), pp. 1–9.

[78] Alexander H. van Zuijlen, Aukje de Boer, and Hester Bijl. "Higher-order time integration through smooth mesh deformation for 3D fluid-structure interaction simulations". In: *J. Comput. Phys.* 224.1 (2007), pp. 414–430. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2007.03.024.