Max Planck Institute for the Science of Light

Artificial Scientist Lab

# Differometor: A Differentiable Interferometer Simulator for the Computational Design of Gravitational Wave Detectors

Jonathan Klimesch, Mario Krenn

# Abstract

We present Differometor, a new differentiable frequency domain interferometer simulator implemented in Python using the JAX framework. Differometor's implementation closely follows the established FINESSE simulator and offers functionality to simulate plane waves in quasi-static, user-specified setup configurations including quantum noise calculations and optomechanical effects. JAX's GPU support and just-in-time compilation ensure fast runtimes, while its automatic differentiation feature enables gradient-based optimizations that can easily support the large-scale digital discovery of novel gravitational wave detectors. Differometor is verified against FINESSE simulations, demonstrating close agreement in strain sensitivity curves of large interferometer setups. Differometor represents a powerful tool for state-of-the-art AI-driven design of novel fundamental physics experiments.

# Contents

# Chapter 1

# Introduction

Gravitational wave detectors, like the Advanced Laser Interferometer Gravitational-Wave Observatory (aLIGO) [1], have opened a new window to observe the universe, independent of electromagnetic waves, neutrinos or massive particles, enabling a new era of multi-messenger astrophysics [2]. Most of these detectors use laser light to sense spacetime and to transform resulting light phase changes into intensity variations by making use of Michelson interferometers [3].

Conventionally, such detectors are designed by computer-aided human intuition and experience guided by mathematical ingenuity. However, when considering the design of new detectors as a combinatorial problem of choosing and placing certain components with specific parameters in a special configuration, it becomes clear that the space of potential designs is big enough to hold large numbers of setups that might have useful properties but have never been explored by human researchers [4, 5].

Through their enormous popularity gain in recent years, AI methods now provide powerful tools to explore this vast search space through automated, digital discovery. To apply these techniques, one requirement is the definition of a well-defined objective function. In the case of gravitational wave detectors, this can be formulated as maximizing a detector's sensitivity to spacetime distortions [6]. To evaluate this objective function in a way that doesn't involve the physical realization of all possible detector blueprints, a high-performance physical simulator is another core requirement.

In the field of gravitational wave detectors, multiple such simulators exist. A prominent example is the frequency domain interferometer simulation software Finesse [7, 8, 9]. This Python and NumPy [10] based simulator was used in recent work on digital discovery of gravitational wave detectors to evaluate optimized detector setups according to their resulting strain sensitivity curves [6]. To do this using gradient-based optimization, the simulator had to be executed millions of times which, despite the high performance of the simulator, resulted in more than one million CPU hours of computational cost. This presents a major bottleneck for large-scale AI-driven exploration tasks.

To reduce this computational cost, the differentiability of the simulator is a crucial requirement for fast gradient-based optimizations. Simulators like FINESSE do not support this and allow only numerical approximation of the gradients which requires multiple simulation runs for each iteration. In addition, existing simulators don't support GPU execution and pose challenges in combining them with modern AI methods such as neural-network-based surrogate models.

Addressing these limitations, we present Differometor, a new differentiable frequency domain interferometer simulator specifically designed for the digital discovery of gravitational wave detectors. Differometor closely follows the implementation of the FINESSE simulator and offers plane-wave propagation through quasi-static, user-specified setup configurations including quantum noise calculations and optomechanical effects. Differometor is implemented in Python using the JAX framework [11]. JAX provides GPU support, automatic differentiation and just-in-time (JIT) compilation, enabling fast runtimes and gradient-based algorithms. Differometor makes efficient use of all of these features and therefore presents a promising solution to current limitations of large-scale digital discovery.

In chapter 2, we first discuss the necessary physical background of gravitational wave and interferometer-based detectors and give an introduction of simulation techniques that we use in Differometor. We then provide more details about recent work on digital discovery of gravitational wave detectors and introduce Differometor. In chapter 3 we describe the different subsystems of Differometor and benchmark it against FINESSE simulations, demonstrating performance advantages while delivering accurate results. In chapter 4 we then apply Differometor to simple digital discovery optimizations with a simplified aLIGO setup and demonstrate its scalability and advantages when running gradient-based optimizations. We conclude in chapter 5 by outlining future development plans.

# Chapter 2

# Background

This chapter provides an introduction to the relevant theory and existing work behind interferometer simulators, as well as the broader goal of discovering new gravitational wave detectors. In section 2.1 we briefly discuss sources of gravitational waves and their effect on test masses. In section 2.2 we then describe interferometer techniques suitable to detect these effects and give a short overview of existing gravitational wave detectors. Section 2.3 introduces the necessary theory and conventions for interferometer simulation and optical modeling, which will be needed for the implementation of our differentiable interferometer simulator in chapter 3. We also list existing interferometer simulation software and introduce FINESSE, which is used as the template for our differentiable simulator. Finally, in section 2.4, we list existing work in the field of AI for science, concentrate on one digital discovery approach for gravitational wave detectors and provide a brief introduction to differentiable programming.

## 2.1 Gravitational Waves

In 1916, Albert Einstein proposed the existence of gravitational radiation as one of the important consequences of his general theory of relativity [12]. The existence of this radiation was then demonstrated by the discovery of the binary pulsar system PSR B1913+16 by Hulse and Taylor [13] and the observation of its energy loss by Taylor and Weisberg in 1982 [14]. On September 14, 2015, a century after Einstein's fundamental prediction, the Laser Interferometer Gravitational-Wave Observatory (LIGO) for the first time directly observed GW150914, a gravitational wave signal generated by the merger of a binary black hole [15].

As an approximation, the emission of gravitational waves can be expressed by defining a gravitational analog to the quadrupole moment of electromagnetic radiation [16]. As for electromagnetic waves, there are different classes of astrophysical systems which emit gravitational waves across a broad frequency spectrum ranging over more than 14 orders of magnitude from $10^{-10}$ Hz to more than $10^4$ Hz [17]. Lower frequency ranges are

populated with gravitational waves emitted from (super-)massive black hole inspiral and mergers, e.g. from remnants of the earliest stars in the universe [18]. Extreme-mass-ratio inspirals like compact objects captured by supermassive black holes emit gravitational waves from around $10^{-4}$ Hz to $10^{-2}$ Hz. The higher frequency ranges are filled by waves from compact binary inspirals and mergers (e.g. black holes or neutron stars from $10^{-4}$ to 5000 Hz) [19], supernovae explosions and pulsars (1 Hz to 1000 Hz) [20, 21] and postmerger events ($10^2$ Hz to $10^3$ Hz) [22].

For the purpose of this thesis, we constrain our understanding of gravitational waves to their actions on test particles in some region of spacetime, as this will be important to simulate their effects. We only sketch the derivation of these actions based on the lecture notes by Blandford and Thorne [23] and refer to these notes for a more formal treatment. Using *Linearized Theory* (idealizing gravitational waves as plane-fronted and propagating through flat spacetime) and introducing coordinates with a small deviation from flat spacetime, the corresponding metric can be written as

$$g_{\alpha\beta} = \eta_{\alpha\beta} + h_{\alpha\beta}, \quad \text{with} \quad |h_{\alpha\beta}| \ll 1 \tag{2.1}$$

where $\eta_{\alpha\beta}$ is the Minkowski metric and $h_{\alpha\beta}$ is the waves' *metric perturbation*. Rewriting this metric perturbation in Lorenz gauge and orienting the axes of the coordinates so that the waves are planar and propagate in z-direction, one can derive a trace-reversed metric perturbation as a solution to the flat-space wave equation. Specializing the gauge further into the so-called *transverse-traceless gauge* or *TT gauge* reduces the independent components of this metric perturbation solution to only two nonzero components $h_{xx} = -h_{yy}$ and $h_{xy} = +h_{yx}$. These two components are associated with two polarization states for the waves ($+$ and $\times$) and are commonly written as

$$h^{\mathrm{TT}}_{xx} = -h^{\mathrm{TT}}_{yy} = h_+(t-z), \quad h^{\mathrm{TT}}_{xy} = +h^{\mathrm{TT}}_{yx} = h_\times(t-z). \tag{2.2}$$

Now we consider a circular ring of test particles that floats freely in space and is static before the gravitational waves pass. Choosing a local Lorentz frame with a reference particle at the ring's center, the waves produce a coordinate displacement $\delta x^j$ described by the displacement vector $\zeta^j = x^j + \delta x^j$ between the reference particle and some other particle with spatial coordinates $x^j$. Inserting this displacement into the local-Lorentz-frame variant of the equation of geodesic deviation, one can derive the *gravitational-wave tidal acceleration* which moves particles back and forth and is described by

$$\delta x^j = \frac{1}{2} h^{\mathrm{TT}}_{jk} x^k. \tag{2.3}$$

This is analogous to the Newtonian tidal acceleration that causes the moon to raise tides in Earth's oceans. We can now insert the two components of the metric perturbation and obtain the final displacements of the ring's particles from the center:

$$\begin{aligned} \delta x &= \frac{1}{2} h_+ x \quad \delta y = -\frac{1}{2} h_+ y, \\ \delta x &= \frac{1}{2} h_\times y \quad \delta y = \frac{1}{2} h_\times x. \end{aligned} \tag{2.4}$$

This shows that the ring's particles are undisturbed in the z-direction, but get deformed into an ellipse, squashed along one axis by the same amount as it is stretched in the other. The area of the ellipse is always preserved. Fig. 2.1 shows this effect on a ring of particles for the + polarization.

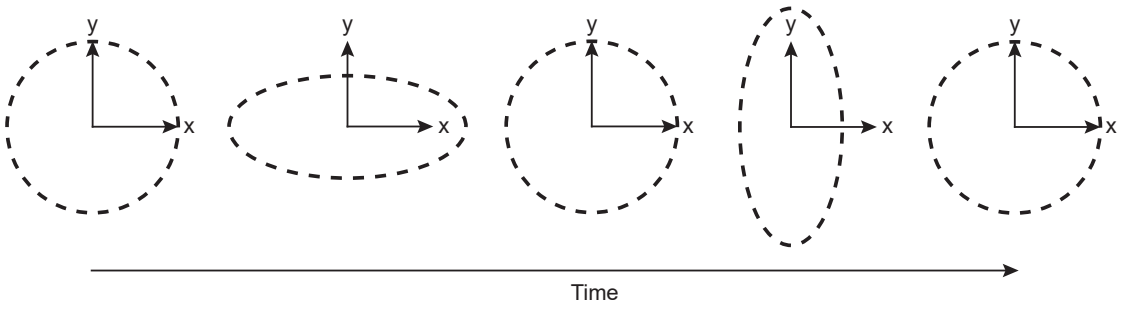


Figure 2.1: Effect of $h_+$ gravitational waves propagating in the z-direction on a ring of particles. While $h_+ > 0$, the ring gets stretched along the x axis and squashed along the y axis. During $h_+ < 0$ the effect is in the reverse direction.

By defining a gravitational analog to the quadrupole moment of electromagnetic radiation and calculating the radiated field from the temporal variation of this quadrupole moment, it is possible to derive an approximation for the amplitude of gravitational waves emitted by a binary system in which a pair of point masses orbit in a circle around their common center of mass [16]:

$$|h| \approx \frac{r_{S1} r_{S2}}{r_0 R}. \tag{2.5}$$

Here, $R$ is the distance from the source to the observation point, $r_0$ is the distance between each object and the common center of mass and $r_{Si}$ is the Schwarzschild radius $r_S = 2GM/c^2$ of the point mass $i$ with mass $M$, gravitational constant $G$ and speed of light $c$. $h$ is the actual measurable strain, that is the fractional amount by which e.g. the distances of the particle ring in Fig. 2.1 get modulated. Using representative values such

as two neutron stars in the Virgo cluster at $R \approx 15$ Mpc with masses close to 1.4 times the mass of the sun, which come so close that they almost touch each other at $r_0 = 20$ km, we obtain an estimate of the dimensionless amplitude of the emitted gravitational wave:

$$|h| \approx 1 \times 10^{-21}. \tag{2.6}$$

Detecting this extremely small amplitude or strain presents a challenge that can only be tackled by the most advanced interferometer techniques described in the next section.

## 2.2 Interferometer Techniques for Gravitational Wave Detection

Here, we will first introduce the basic detection principle of interferometric gravitational wave detectors in section 2.2.1 and then give an overview of past, present and future detectors in section 2.2.2.

### 2.2.1 Detector Principles

A gravitational wave detector measures the gravitational wave induced spacetime perturbations described in section 2.1. The most popular design to do this is to use laser light to sense spacetime and transform resulting light phase changes into intensity variations by making use of interference [3].

First, we consider the effect of a gravitational wave on an optical measurement of the spatial distance. For this we mainly follow Jun Mizuno's derivation from [24]. We consider two free masses separated by length $L$. Light travelling between these two masses has the null proper time

$$(ds)^2 = g_{\mu\nu} dx^\mu dx^\nu = -(cdt)^2 + g_{ij} dx^i dx^j = 0 \quad (i, j = 1, 2, 3). \tag{2.7}$$

Concentrating on gravitational waves with the $h_+$ polarization described by equation 2.2, we get:

$$(ds)^2 = -(cdt)^2 + (1 + h_+)\,(dx)^2 + (1 - h_+)\,(dy)^2 + (dz)^2 = 0. \tag{2.8}$$

If we now look at light travelling between these two masses in y-direction, we insert $dx = dz = 0$ and find:

$$\left(\frac{dy}{dt}\right)^2 = \frac{c^2}{1-h_+}. \tag{2.9}$$

For our optical measurement, we are interested in the phase change that is induced by the gravitational wave. We can calculate that phase change from the round trip time $t_r$ of light with angular frequency $\omega_0$ between these two masses using $\phi(t) = \omega_0 t_r$. We can calculate the round trip time using

$$\begin{aligned} 2L &= \int_0^L dy - \int_l^0 dy = \int_{t-t_\mathrm{r}}^{t} \frac{dy}{dt} dt \\ &= \int_{t-t_\mathrm{r}}^{t} \frac{c}{\sqrt{1-h_+(t)}} dt = c\int_{t-t_\mathrm{r}}^{t} \left\{1 + \frac{1}{2}h_+(t) + O\left(|h_+|^2\right)\right\} dt \\ t_\mathrm{r} &\simeq \frac{2L}{c} - \frac{1}{2}\int_{t-t_\mathrm{r}}^{t} h_+(t)dt \\ &\simeq \frac{2L}{c} - \frac{1}{2}\int_{t-2L/c}^{t} h_+(t)dt \quad (\text{ for } |h_+| \ll 1)\,. \end{aligned} \tag{2.10}$$

The light along the x-axis experiences the same effect, but with a different sign, which again shows the typical squeezing and stretching behavior already described in section 2.1. The phase change is thus given by

$$\varphi(t) = \omega_0 t_\mathrm{r} \simeq \frac{2\omega_0 L}{c} \pm \frac{\omega_0}{2}\int_{t-2L/c}^{t} h_+(t)dt. \tag{2.11}$$

We now follow Bond et al. [25] and assume a gravitational wave signal given by:

$$h(t) = h_0 \cos\left(\omega_g t + \varphi_g\right). \tag{2.12}$$

Here, $h_0$ is the amplitude of the gravitational wave, calculated for example values in equation 2.6. Looking for the phase change of a one-wave trip between the two masses, we get

$$\varphi(t) = -\frac{\omega_0 L}{c} \mp \frac{\omega_0}{2}\int_{t-L/c}^{t} h(t)dt = -\frac{\omega_0 L}{c} \mp \frac{\omega_0 h_0}{2}\left[\frac{1}{\omega_g}\sin\left(\omega_g t + \varphi_g\right)\right]_{t-L/c}^{t} \tag{2.13}$$

which we can simplify using trigonometric identities:

$$\varphi(t) = -\frac{\omega_0 L}{c} \mp \frac{\omega_0 h_0}{\omega_g} \cos\left(\omega_g t + \varphi_g - \omega_g \frac{L}{2c}\right) \sin\left(\omega_g \frac{L}{2c}\right). \tag{2.14}$$

This result will be used in section 2.3.3 to simulate the effect of a gravitational wave. We can transform such phase changes into intensity variation by making use of two coherent light waves because the amplitude of their superposition depends on their relative phase. This interference of light has been common knowledge since a couple of centuries and is manifested in a multitude of interferometers of various designs.

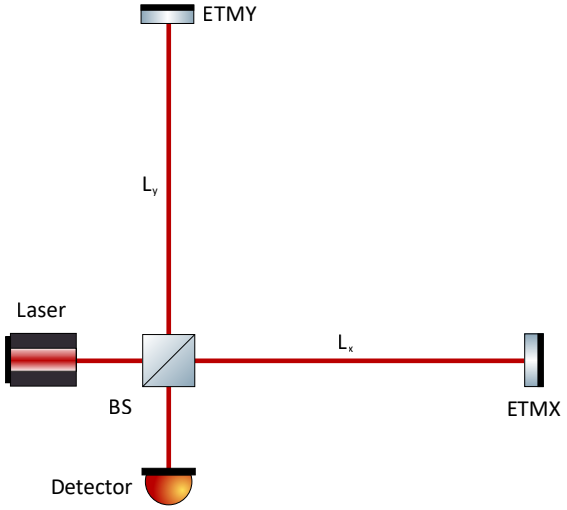


Figure 2.2: A Michelson interferometer with a central beam splitter and one end test mass mirror on each arm.

The most popular one for gravitational wave detection is the Michelson interferometer shown in Fig. 2.2 [25]. In this design, a central beam splitter is used to split the laser beam into two waves with equal amplitudes and send them into two perpendicular interferometer arms with lengths $L_x$ and $L_y$. Two highly reflective mirrors at the end of these arms, often referred to as end test masses (ETM), reflect the two beams back to the beam splitter, where they recombine and are detected by a photodetector in the signal port. The lengths of the interferometer arms are tuned so that the optical paths of the two light beams are equal to each other and that the beams interfere destructively at the photodetector [3]. This operation mode is called *dark fringe* as it leaves the photodetector unilluminated. If one of the end test masses then gets displaced by a gravitational wave, the difference in arm lengths is non-zero: $\delta L = L_x - L_y \neq 0$. Writing the laser beam before the beam splitter as

$$E_{\text{laser}}(t) = E_0 \cos(\omega_0 t) \tag{2.15}$$

and the reflected beams in the two perpendicular arms as

$$E_{x,y}(t) = -\frac{E_0}{\sqrt{2}} \cos(\omega_0 t - 2\omega_0 L_{x,y}/c), \tag{2.16}$$

then the output light field at the photodetector can be written as:

$$E_{\text{dark}}(t) = \frac{E_x^{\text{out}}(t) - E_y^{\text{out}}(t)}{\sqrt{2}} = E_0 \sin\frac{\omega_0 \delta L}{c} \sin(\omega_0 t - \omega_0 [L_x + L_y]/c). \tag{2.17}$$

We can calculate the resulting intensities by averaging the field amplitudes over many oscillation periods via $\mathcal{I} \propto \overline{E^2}$:

$$\mathcal{I}_{\text{dark}}(\delta L/\lambda_0) = \frac{\mathcal{I}_0}{2}\left(1 - \cos 4\pi \frac{\delta L}{\lambda_0}\right). \tag{2.18}$$

The Michelson interferometer tuned to operate at the dark fringe therefore has a sensitivity proportional to $(\delta L/\lambda_0)^2$. For small differential displacements $\delta L \ll \lambda_0$, this yields extremely weak light power on the photodetector. Therefore, interferometers are usually slightly detuned from the dark fringe in order to ensure a linear instead of quadratic dependence on the displacement.

To detect the extremely small spacetime perturbations as approximated in equation 2.6, the most important property of a gravitational wave detector is its sensitivity expressed by the noise-to-signal ratio (NSR) which measures the minimal measurable amplitude of a gravitational wave [25]:

$$\text{NSR} = \frac{\sqrt{S_{\text{P}}}}{T_{gw \to P}}. \tag{2.19}$$

Here, $T_{gw \to P}$ is the transfer function from a gravitational wave signal to the output photodiode. It outputs Watts per unit strain $h$ at a certain signal frequency $\omega_{gw}$ and describes how the signal gets transformed when passing through the interferometer to the detector. The calculation of this transfer function via so-called coupling matrices is described in section 2.3.2. $S_{\text{P}}$ is the power spectral density (PSD) of the noise present in the detector, measured in watts per unit frequency. The sensitivity of a gravitational wave detector is limited by different noise terms [26].

Thermal noise includes e.g. thermal motion of the test mass suspensions as well as Brownian motion of the optic dielectric coatings. Seismic noise describes ground motion induced movement of the test masses and is mitigated by suspending the test masses from quadruple pendulum chains. Newtonian noise stems from gravitational coupling of the test masses to fluctuating mass density fields, e.g. produced by seismicity and atmospheric pressure fluctuations. In addition, there is laser frequency and intensity noise, different control noises, residual gas noise and others.

Once classical noise sources have been mitigated, quantum noise imposes a fundamental limit to the interferometer sensitivity [27, 28, 29, 30]. This quantum noise appears in two forms, *shot noise* and *radiation pressure noise*. Shot noise dominates the high-frequency region of the sensitivity spectrum and arises from statistical fluctuations in the arrival time of photons at the detector. Radiation pressure noise dominates at low frequencies as it is a displacement noise from amplitude fluctuations of the light field in the interferometer arms, which generate a motion of the optics. The inset in Fig. 2.3 shows simulated strain sensitivity curves which contain these two quantum noise types.

Shot noise is usually reduced by two techniques, an increase in laser power or the use of squeezed vacuum injected into the dark port of the interferometer [26]. Squeezing vacuum noise is the process of decreasing the uncertainty in either phase or amplitude quadrature of a light field. Due to the uncertainty principle, squeezing the phase quadrature to reduce shot noise leads to anti-squeezing in the amplitude quadrature which in turn raises radiation pressure noise. However, a broadband reduction of both types of quantum noise can be achieved by frequency dependent squeezing in which low frequency vacuum noise is amplitude squeezed to reduce radiation pressure while high frequency noise is phase squeezed to reduce shot noise [31].

More details about shot noise and radiation pressure noise and how to simulate them is described in sections 2.3.5 and 2.3.6.

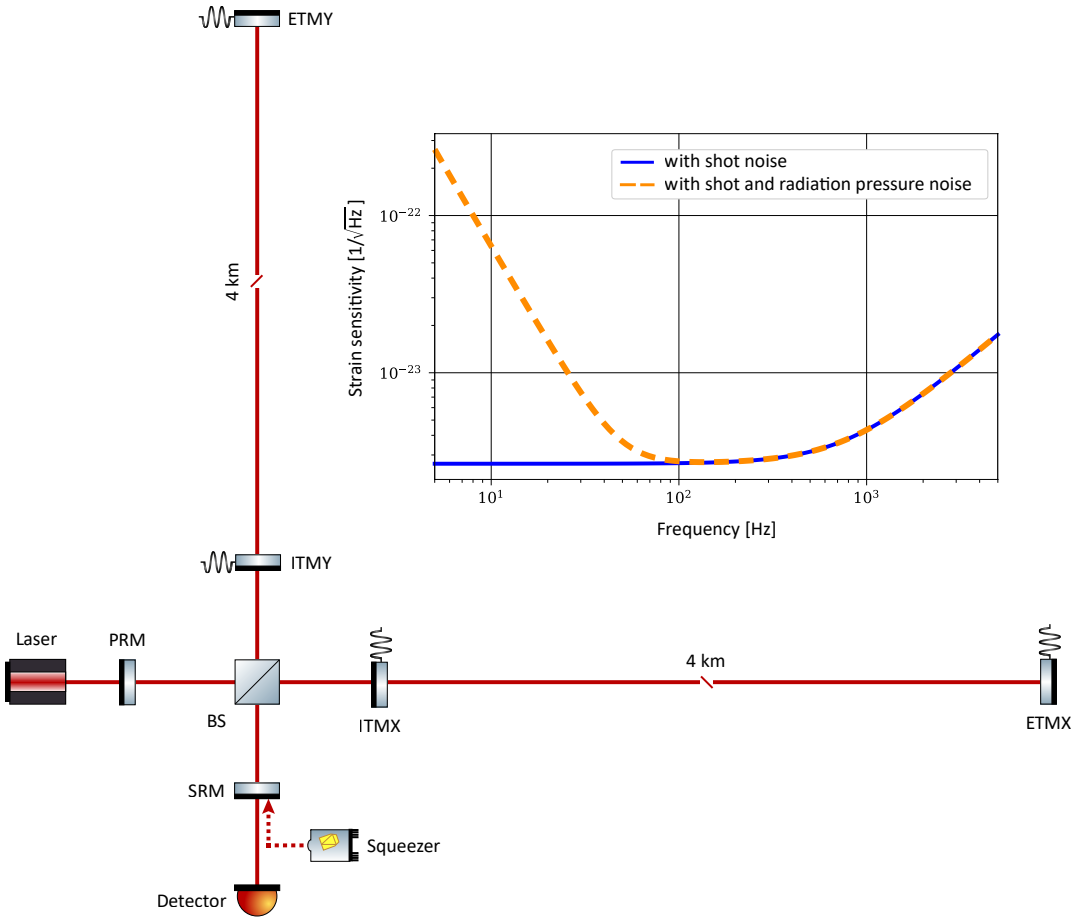


Figure 2.3: Simplified optical layout of the aLIGO observatories that will be used in optimizations in later sections. These detectors are dual-recycled, Fabry-Pérot, Michelson interferometers with suspended test masses and a squeezed vacuum source to reduce quantum noise. Inset: The simulated strain sensitivity for this setup, once with shot noise only and once taking into account the radiation pressure with the suspended mirrors. Shot noise dominates the high frequencies while radiation pressure noise dominates the low frequency regions. The sensitivity curves were calculated using FINESSE and the layout was taken from [9].

### 2.2.2 Existing Gravitational Wave Detectors

The first generation of initial detectors in the early 2000s (TAMA300 [32], GEO600 [33], Virgo [34], LIGO [35]) served as a proof-of-concept for second generation detectors (aLIGO [1] and Advanced Virgo [36]) which detected the first gravitational wave in 2015 [15]. Future third generation detectors include the Cosmic Explorer [37, 38] and the Einstein Telescope [39]. The NEMO detector [40] is supposed to complement these

detectors at higher frequencies while the Laser Interferometer Space Antenna (LISA) [41] and atom interferometers like the MAGIS-km [42] fill lower and medium frequency ranges [43].

In section 4.2 we will run optimizations based on a simplified design of the aLIGO detector [9], shown in Fig. 2.3. This detector consists of two laser interferometers around 3000 km apart and is currently conducting its fourth observation run [44]. The core of the two observatories are dual-recycled, Fabry-Pérot, Michelson interferometers where the arms are cavities of 4 km length. The arm cavities enhance the light power circulating in the interferometer arms by constructive interference between the input test masses (ITMs) and end test masses (ETMs) while not increasing the light power in the beam splitter substrate. They also result in an increase of sensitivity for gravitational waves with a frequency within the linewidth of the cavities, but also in a decrease in sensitivity for signals outside their linewidth [45].

Dual-recycling refers to one power-recycling mirror at the input between laser and beam splitter and one signal-recycling mirror at the output port of the beam splitter. Power recycling was first proposed in 1983 [46, 47] and further increases the amount of light circulating within the interferometer while also reducing beam jitter and laser frequency noise. The disadvantage is an increase in laser power in the beam splitter substrate which can lead to higher-optical loss by causing thermal distortions. Signal recycling was first suggested in 1988 [48] and additionally increases the power detected on the photodetector while also increasing the *bandwith* of the detector for the signal sidebands. Here, the bandwidth refers to the frequency at which the differential arm frequency response begins falling off.

Fig. 2.3 also shows the optics suspensions of the cavity mirrors affecting the optomechanics of the system which is explained in more detail in section 2.3.6. aLIGO also makes use of squeezers to reduce quantum noise as explained in section 2.2.1. Not shown in Fig. 2.3 is the filter cavity at the squeezer which enables aLIGO to perform frequency dependent squeezing. Also not shown in Fig. 2.3 are the input mode cleaner used for stabilizing the laser frequency, intensity and spatial mode content, the output mode cleaner used for intensifying only the main interferometer gravitational wave signal, the seismic isolation systems, the filter cavity that enables frequency-dependent squeezing and the data acquisition systems. We refer to [44] for a full review of aLIGO components.

Similar to the simulated sensitivity curve in Fig. 2.3, the aLIGO detector targeted a detection band from 10 Hz to 7000 Hz and had a design sensitivity reaching $10^{-23}/\sqrt{\text{Hz}}$ in the most sensitive frequency ranges.

## 2.3 Interferometer Simulation and Optical Modeling

In this section, we will provide the theoretical background necessary to implement the differentiable interferometer simulator in the next chapter. First, we define light fields, describe their interaction with interferometer components and introduce common conventions in interferometer simulation. In further subsections, light field modulation and readout techniques are described and an introduction to the sideband formulation of quantum noise and to optomechanics is given. For this, we mainly follow the reviews by Bond et al. [25] and by Danilishin et. al. [3] as well as the manuals of FINESSE 2 [8] and FINESSE 3 [9].

### 2.3.1 Light Fields

Starting from Maxwell's equations, we can formulate the homogeneous wave equation whose solutions are plane waves [49]. For our purposes we can ignore the polarization and focus on the electric component expressed as scalar waves. Using complex notation and assuming propagation through vacuum, these can be described as

$$E = E_0 \exp(\mathrm{i}(\omega t - kD)) = E_0 \exp(\mathrm{i}(\omega t + \varphi)). \tag{2.20}$$

Here, $E_0$ is the light field amplitude, $\omega = 2\pi f$ is the angular frequency of the wave, $t$ is time, $k = \omega/c$ is the wave number in vacuum, $c$ is the speed of light, $D$ is the distance that the wave travels and $\varphi$ a constant phase term.

We are interested in implementing a frequency domain interferometer simulator and can assume that the simulated interferometer is in a steady state with solutions independent of time. In this case, equation 2.20 can be simplified to

$$E = E_0 \exp(-\mathrm{i}kD) = E_0 \exp(i\varphi). \tag{2.21}$$

The next section explains how these light fields interact with interferometer components such as mirrors, beam splitters and the spaces in between.

### 2.3.2 Coupling Matrices of Optical Components

Again starting from Maxwell's equations, we can formulate continuity conditions for electromagnetic waves at interfaces. From these, we can derive the laws of reflection and refraction together with the Fresnel equations and amplitude coefficients for reflection and transmission r and t, with $0 \leq r^2, t^2 \leq 1$. For a detailed derivation, we refer to the introductory book by Nolting [49].

For our purposes we use the coefficients $r$ and $t$ to describe the interaction of light fields with mirrors and beam splitters as main interferometer components. For that, we make use of a simplifying theoretical model of these components, viewing them as single, flat, symmetric optical surfaces while realistic models would include two specifically shaped optical surfaces with a substrate in between. In short, we describe the interactions between light fields and interferometer components using linear coupling coefficients. This approach already allows us to simulate simplified interferometer layouts while it is in principle expandable with more complex and realistic phenomena.

A mirror is a linear system with two input and two output ports as shown in Fig. 2.4. The field amplitudes of the light fields at the output ports can be written as

$$\begin{aligned} E_2 &= rE_1 + \mathrm{i}tE_3 \\ E_4 &= rE_3 + \mathrm{i}tE_1 \end{aligned} \tag{2.22}$$

or in matrix form as

$$\begin{pmatrix} E_2 \\ E_4 \end{pmatrix} = \begin{pmatrix} \mathrm{i}t & r \\ r & \mathrm{i}t \end{pmatrix} \begin{pmatrix} E_3 \\ E_1 \end{pmatrix}. \tag{2.23}$$

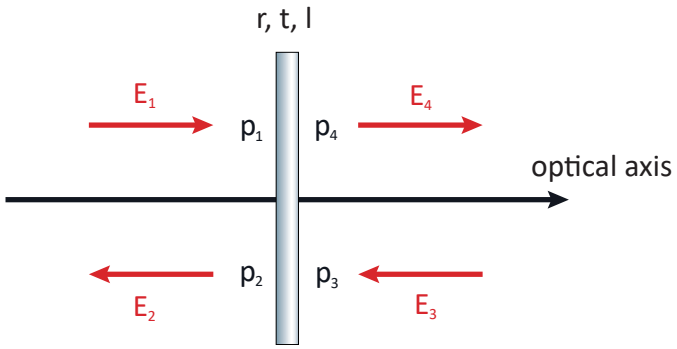


Figure 2.4: A mirror component with two input ($p_1$, $p_3$) and two output ports ($p_2$, $p_4$), coefficients for reflection and transmission $r, t$ and loss $l$.

This is commonly called the *coupling matrix* for field amplitudes at a mirror. This definition also includes a potential loss by defining loss $0 < L < 1$, reflectivity $R = r^2$ and transmissivity $T = t^2$ and constraining these three parameters through the relation $L + R + T = 1$.

The $\pi/2$ phase shift upon transmission follows a common convention for the analysis of modern optical systems. For a detailed explanation of this convention, we refer to

section 2.4 of the review by Bond et al. [25]. In short, the absolute phase change upon interaction with an optical surface depends on the applied coating and is generally not known. However, because the positions of optical components are not known to sub-wavelength precision, this absolute phase change is typically not of interest. Instead, the relative phase relation between incoming and outgoing beams is of importance and constrained by the following expression derived from the fundamental principle of power conservation:

$$\frac{1}{2}\left(\varphi_{r1}+\varphi_{r2}\right)-\varphi_t=(2N+1)\frac{\pi}{2}. \tag{2.24}$$

Here, $\varphi_{r1}$ and $\varphi_{r2}$ are the phase changes upon reflection on both sides of the mirror and $\varphi_t$ is the phase change upon transmission. $N$ is an integer that can be chosen arbitrarily and will be set to $N=-1$ throughout this text. This constraint on the relative phase relation allows for the choice of any set of phase changes on reflection and transmission that fulfils it. Thus, the convention is to choose $\phi_t=\pi/2$ and $\varphi_{r1}=\varphi_{r2}=\varphi_r=0$ which is convenient as it allows the definition of mirrors and beam splitters without a front and back face.

Similar to the coupling matrix for a mirror, we then define the coupling matrix for a beam splitter with four input and four output ports (see Fig. 2.5) as

$$\begin{pmatrix} E_2 \\ E_4 \\ E_6 \\ E_8 \end{pmatrix} = \begin{pmatrix} 0 & r & it & 0 \\ r & 0 & 0 & it \\ it & 0 & 0 & r \\ 0 & it & r & 0 \end{pmatrix} \begin{pmatrix} E_1 \\ E_3 \\ E_5 \\ E_7 \end{pmatrix}. \tag{2.25}$$

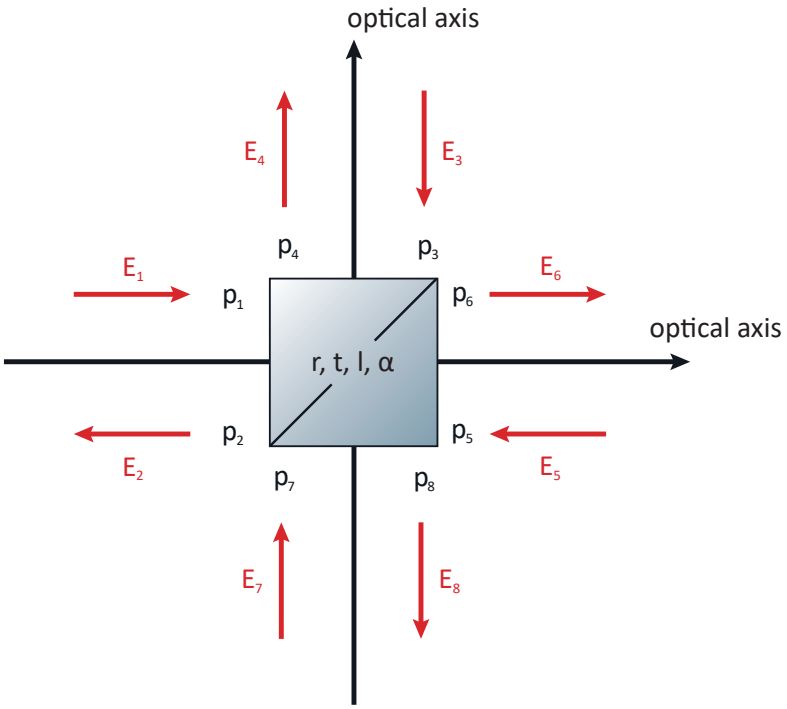


Figure 2.5: A beam splitter component with 4 input and 4 output ports, coefficients for reflection and transmission $r, t$, loss $l$ and angle of incidence $\alpha$.

Furthermore, when freely propagating through a medium with refraction index $n$ and distance $D$, a light field accumulates an additional phase. According to equation 2.21, this additional phase can be expressed as $k_n D$ where the wave number $k_n$ is defined as $\omega/v_n$ with $\omega$ as angular frequency and $v_n$ as phase velocity of the wave in a material with refractive index $n$. The refractive index is defined as $n = c/v_n$, which means we can rewrite the accumulated phase over distance $D$ as $knD$ with $k$ as the wave number in vacuum [49]. Now we can formulate the coupling matrix for light field propagation in a material with refractive index $n$ over distance $D$ as

$$\begin{pmatrix} E_2 \\ E_4 \end{pmatrix} = \begin{pmatrix} \exp(-\mathrm{i}knD) & 0 \\ 0 & \exp(-\mathrm{i}knD) \end{pmatrix} \begin{pmatrix} E_3 \\ E_1 \end{pmatrix}. \tag{2.26}$$

In order to combine coupling matrices to describe an optical system with multiple components, there are two commonly used methods. The first method is to construct an *interferometer matrix* with one equation for each light field amplitude. For example, we can rewrite the coupling matrix for a mirror from equation 2.23 as a full interferometer matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -r & 1 & -it & 0 \\ 0 & 0 & 1 & 0 \\ -it & 0 & -r & 1 \end{pmatrix} \begin{pmatrix} E_1 \\ E_2 \\ E_3 \\ E_4 \end{pmatrix} = \begin{pmatrix} E_{1i} \\ 0 \\ E_{3i} \\ 0 \end{pmatrix} = M_{\text{interferometer}}\ \mathbf{E}_{\text{sol}} = \mathbf{E}_{\text{input}} \tag{2.27}$$

where $\mathbf{E}_{\text{input}}$ has non-zero values for any input fields and $\mathbf{E}_{\text{sol}}$ is the solution vector that provides information about all light field amplitudes in the entire system after solving. If we have multiple components, we combine them into an interferometer matrix by placing their matrices along the diagonal and connecting them with off-diagonal space connector entries. A cavity setup with two mirrors and a space in between as shown in Fig. 2.10 would then result in the interferometer matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -r_1 & 1 & -it_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -e^{-iknD} & 0 & 0 \\ -it_1 & 0 & -r_1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -e^{-iknD} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -r_2 & 1 & -it_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -it_2 & 0 & -r_2 & 1 \end{pmatrix} \begin{pmatrix} E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \\ E_6 \\ E_7 \\ E_8 \end{pmatrix} = \begin{pmatrix} E_{1i} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ E_{7i} \\ 0 \end{pmatrix} \tag{2.28}$$

where $E_{1i}$ and $E_{7i}$ are potential input fields. Note that the interferometer matrices for mirror 1 and mirror 2 are simply inserted unchanged and connected via the space connector entries.

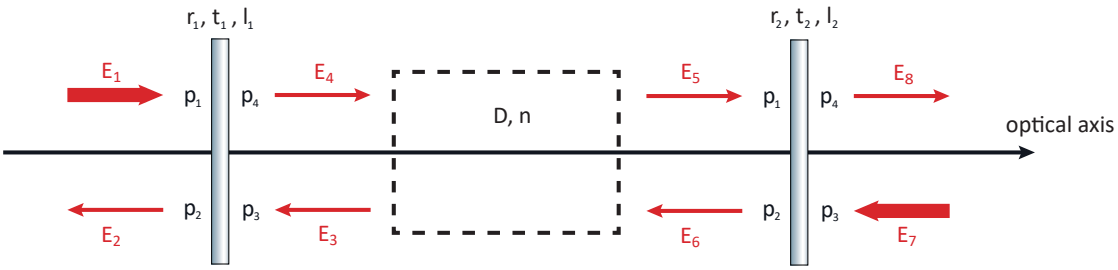


Figure 2.6: A cavity setup with 2 mirrors and a space in between. It is fully described by the interferometer matrix in equation 2.28. $E_1$ and $E_7$ are marked as possible inputs to the setup.

The second method to describe a full system of coupling matrices follows a more compact and sequential approach [50]. It requires reordering the coupling matrices so that light

fields on the same side of the optical component are also on the same side of the system of equations. The coupling matrix for a mirror (equation 2.23) would then be rewritten into

$$\begin{pmatrix} E_1 \\ E_2 \end{pmatrix} = \frac{i}{t} \begin{pmatrix} -1 & r \\ -r & r^2 + t^2 \end{pmatrix} \begin{pmatrix} E_4 \\ E_3 \end{pmatrix}. \tag{2.29}$$

Following this approach one could then express a cavity setup by multiplying the respective component matrices:

$$M_{\text{cav}} = M_{\text{mirror 1}} \times M_{\text{space}} \times M_{\text{mirror 2}}. \tag{2.30}$$

This yields another compact 2x2 matrix that can be used for the calculation of the output fields. The disadvantage of this method is the lack of information about the light fields within the optical system, e.g. at the right ports of mirror 1 and left ports of mirror 2. In order to constrain the light power impinging on component ports, one would need this information which makes this method only applicable for certain kinds of analysis.

Storing the distance $D$ from the space coupling matrix in equation 2.26 directly as a floating point number leads to unacceptable rounding errors. On the one hand there are parameters (e.g. the resonance condition inside an optical cavity) which depend on the distance modulo the laser wavelength. Thus, these parameters often depend on length differences on the order of 1 $\mu$m. On the other hand, properties like the finesse of a cavity or the propagation of sideband fields depend on the absolute lengths. Dealing with such different magnitude requirements by storing the distance as a floating point number often leads to inaccuracies. A common convention to cope with this difficulty is to split distances into a macroscopic length $L$ defined as the multiple of a constant default wavelength $\lambda_0$ and a microscopic tuning $T$ so that $D = L + T$ [51]. $\lambda_0$ can thereby be understood as the laser wavelength in vacuum or has to be chosen arbitrarily if multiple light fields with different frequencies are simulated. A phase difference caused by an additional distance $D$ is given by

$$\varphi = -kD. \tag{2.31}$$

With the default wavelength $\lambda_0$ we define $\omega_0 = 2\pi c/\lambda_0$ and $k_0 = \omega_0/c$. We can then express any given wavelength $\omega$ relative to the default wavelength $\omega_0$ by defining $\omega = \omega_0 + \Delta\omega$ and the wavenumber as $k = k_0 + \Delta k$. Thus, equation 2.31 becomes

$$-\varphi = kD = \frac{\omega_0 L}{c} + \frac{\Delta\omega L}{c} + \frac{\omega T}{c} \tag{2.32}$$

where the first term is always a multiple of $2\pi$ as $L$ is a multiple of $\lambda_0$. Thus, the first term is equivalent to zero. Now we can define the dimensionless tuning:

$$\phi = \frac{\omega_0 T}{c}. \tag{2.33}$$

Equation 2.32 can then be rewritten as

$$-\varphi = kD = \frac{\Delta\omega L}{c} + \phi\frac{\omega}{\omega_0}. \tag{2.34}$$

The tuning $\phi$ can then be treated as a parameter of optical components and represents their microscopic displacement. The length $L$ on the other hand is the macroscopic length of a space between components.

With this new definition of macroscopic length and microscopic tuning, the coupling matrices have to get updated. Reflected and transmitted light fields now get affected by the tuning distance, so there is an additional phase accumulation described by the second term in equation 2.34. For reflected light fields, the tuning induced phase shifts are

$$\varphi_{r1} = 2\phi n_1 \frac{\omega}{\omega_0} \qquad \varphi_{r2} = -2\phi n_2 \frac{\omega}{\omega_0}. \tag{2.35}$$

The refractive index comes from defining the tuning with $c$ as vacuum phase velocity while in general there could be materials with different refractive indices on both sides. With $N = -1$, equation 2.24 then gives the tuning induced phase shift for transmitted light, resulting in

$$\varphi_t = \frac{\pi}{2} + \frac{\varphi_{r1} + \varphi_{r2}}{2} \tag{2.36}$$

Fig. 2.7 shows a tuned mirror. Because of the additional phase accumulation the mirror's coupling matrix (equation 2.23) has to be rewritten as

$$\begin{pmatrix} E_2 \\ E_4 \end{pmatrix} = \begin{pmatrix} t\ \mathrm{e}^{\mathrm{i}\varphi_t} & r\ \mathrm{e}^{\mathrm{i}\varphi_{r1}} \\ r\ \mathrm{e}^{\mathrm{i}\varphi_{r2}} & t\ \mathrm{e}^{\mathrm{i}\varphi_t} \end{pmatrix} \begin{pmatrix} E_3 \\ E_1 \end{pmatrix}. \tag{2.37}$$

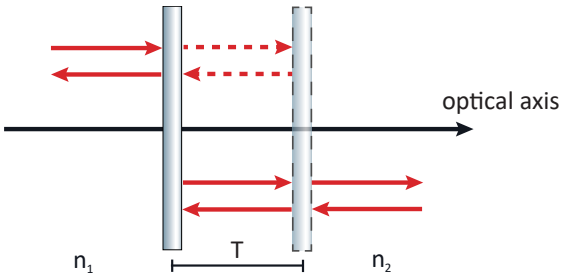


Figure 2.7: A mirror with tuning $T$. The dashed and transparent mirror marks the original mirror position while the solid mirror marks the position after tuning. Both sides can have different refractive indices $n_1$ and $n_2$. Dashed red arrows mark the original light paths, solid red arrows the new light paths, which are shorter for beams from the left and longer for beams from the right, resulting in phase differences for the impinging light fields.

The beam splitter coupling matrix has to be updated in a similar way. A tuned beam splitter is shown in Fig. 2.8. The only difference to a tuned mirror is an additional parameter $\alpha$ specifying the tilt angle relative to the incoming beams on the left side of the beam splitter. In this case, the tuning induced phase shifts for reflected light fields are

$$\varphi_{r1} = 2\phi n_1 \cos\alpha \frac{\omega}{\omega_0} \qquad \varphi_{r2} = -2\phi n_2 \cos\beta \frac{\omega}{\omega_0}. \tag{2.38}$$

where $\beta$ is the tilt angle on the right side of the beam splitter, which can be calculated via Snell's law:

$$n_1 \sin\alpha = n_2 \sin\beta. \tag{2.39}$$

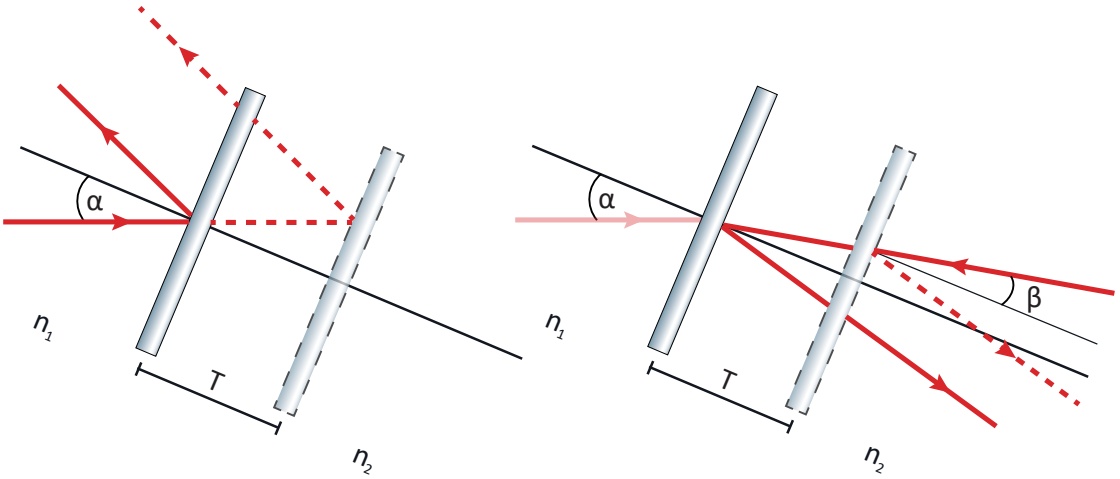


Figure 2.8: A beam splitter with tuning $T$. The dashed, transparent beam splitter marks the original position, the solid beam splitter the position after tuning. Both sides can have different refractive indices $n_1$ and $n_2$. Dashed red arrows mark the original light paths, solid red arrows the new light paths, which are shorter for beams from the left and longer for beams from the right, resulting in phase differences for the impinging light fields.

For the phase change upon transmission, the shift is equivalent to equation 2.36. With this, reflectivity and transmissivity entries of the coupling matrix for the beam splitter have to be updated in the same way as for the mirror coupling matrix:

$$\begin{pmatrix} E_2 \\ E_4 \\ E_6 \\ E_8 \end{pmatrix} = \begin{pmatrix} 0 & r\,\mathrm{e}^{\mathrm{i}\varphi_{r1}} & t\,\mathrm{e}^{\mathrm{i}\varphi_t} & 0 \\ r\,\mathrm{e}^{\mathrm{i}\varphi_{r1}} & 0 & 0 & t\,\mathrm{e}^{\mathrm{i}\varphi_t} \\ t\,\mathrm{e}^{\mathrm{i}\varphi_t} & 0 & 0 & r\,\mathrm{e}^{\mathrm{i}\varphi_{r2}} \\ 0 & t\,\mathrm{e}^{\mathrm{i}\varphi_t} & r\,\mathrm{e}^{\mathrm{i}\varphi_{r2}} & 0 \end{pmatrix} \begin{pmatrix} E_1 \\ E_3 \\ E_5 \\ E_7 \end{pmatrix}. \tag{2.40}$$

For the propagation through a space with refractive index $n$ and macroscopic length $L$, there is the phase factor described by the first term in equation 2.34. The updated coupling matrix for such a space can be written as

$$\begin{pmatrix} E_2 \\ E_4 \end{pmatrix} = \begin{pmatrix} \mathrm{e}^{-\mathrm{i}\frac{\Delta\omega}{c}nL} & 0 \\ 0 & \mathrm{e}^{-\mathrm{i}\frac{\Delta\omega}{c}nL} \end{pmatrix} \begin{pmatrix} E_3 \\ E_1 \end{pmatrix}. \tag{2.41}$$

Combining coupling matrices of different components of an optical system to form an interferometer matrix (e.g. equation 2.28) and then solving the resulting system of equations for the light fields at all component ports enables the propagation of laser light, i.e. the carrier field, through the system. The same mechanism can be used to also propagate signal sidebands which will be introduced in the next section.

### 2.3.3 Light Field Modulation

In interferometer simulation, one usually deals with three different types of light fields. First, there is the laser itself, for example with a frequency of $f \approx 2.8 \cdot 10^{14}$ Hz. Then there are so-called *radio frequency* (RF) sidebands often used for optical readout purposes and with frequencies (offset to the laser frequency) around $f \approx 1 \cdot 10^6$ to $150 \cdot 10^6$ Hz. The *signal* sidebands carry the signal to be measured (e.g. a gravitational wave signal or noise) and have frequencies of $1 - 10000$ Hz.

Sidebands are for example generated by light field modulation. In the frequency domain, such sidebands are just new light fields, which are shifted in frequency with respect to the carrier field. Light power is then divided between the carrier field and the different sideband fields. Here we describe the computation of sideband fields created by phase modulation which will later be used to simulate signal sidebands from gravitational waves and optomechanical effects.

For phase modulation and modulation by a gravitational wave, we follow Bond et al. [45] and start with a light field defined by equation 2.20. We then apply a phase modulation and get:

$$E = E_0 \exp(\mathrm{i}(\omega_0 t + \varphi_0 + \phi(t))), \tag{2.42}$$

where $\omega_0$ and $\varphi_0$ are the angular frequency and the phase of the carrier and $\phi(t)$ is the modulation signal defined by:

$$\phi(t) = m \cos(\Omega t + \varphi_s)\,. \tag{2.43}$$

Here, $m$ is the so-called modulation index and $\varphi_s$ the phase of the modulation signal. We can now expand this modulated light field as a series of Bessel functions of the first kind, $J_k(m)$ to make it explicit that this phase modulation actually creates an infinite number of upper ($k > 0$) and lower ($k < 0$) sidebands around the carrier ($k = 0$):

$$\exp(\mathrm{i}m \cos\varphi) = \sum_{k=-\infty}^{\infty} \mathrm{i}^k J_k(m) \exp(\mathrm{i}k\varphi) \tag{2.44}$$

Bessel functions of the first kind are defined as:

$$J_k(m) = \left(\frac{m}{2}\right)^k \sum_{n=0}^{\infty} \frac{\left(-\frac{m^2}{4}\right)^n}{n!(k+n)!}. \tag{2.45}$$

In order to propagate the resulting sidebands through the optical system, we have to limit them to a reasonable number. Usually the signal amplitude, that is the modulation index $m$, is assumed to be much smaller than 1. With that we can approximate the Bessel functions as

$$J_k(m) = \frac{1}{k!}\left(\frac{m}{2}\right)^k + O\left(m^{k+2}\right) \tag{2.46}$$

and

$$J_{-k}(m) = (-1)^k J_k(m) \tag{2.47}$$

as they decrease rapidly with increasing $k$. This also means that in this case, we only have to take a few sidebands into account. For the carrier at $k = 0$ we approximate equation 2.45 up to the second order in m resulting in

$$J_0(m) = 1 - \frac{m^2}{4}. \tag{2.48}$$

For the sidebands at $k = \pm 1$ we use equation 2.46 and obtain

$$J_{-1}(m) = -\frac{m}{2} \quad J_1(m) = \frac{m}{2}. \tag{2.49}$$

We can now insert the sideband series from equation 2.44 into the modulated light field from equation 2.42:

$$E = E_0 \exp(\mathrm{i}(\omega_0 t + \varphi_0)) \sum_{k=-\infty}^{\infty} \mathrm{i}^k J_k(m) \exp(\mathrm{i}k\varphi). \tag{2.50}$$

By restricting the series to the three Bessel function approximations from equations 2.48 and 2.49 and by replacing $\varphi$ with the cos term from the modulation signal in equation 2.43, we can write the modulated field as [25]:

$$\begin{aligned} E =& E_0\left(1 - \frac{m^2}{4}\right)\exp\left(i\left(w_0 t + \varphi_0\right)\right) \\ &+ E_0\frac{m}{2}\exp\left(i\left(\left(w_0 - \Omega\right)t + \varphi_0 + \frac{\pi}{2} - \varphi_s\right)\right) \\ &+ E_0\frac{m}{2}\exp\left(i\left(\left(w_0 + \Omega\right)t + \varphi_0 + \frac{\pi}{2} + \varphi_s\right)\right) \end{aligned} \tag{2.51}$$

with sideband amplitudes

$$A_{sb} = \frac{m}{2} E_0 \tag{2.52}$$

and sideband phases of

$$\varphi_{sb} = \varphi_0 + \frac{\pi}{2} \pm \varphi_s. \tag{2.53}$$

This describes a carrier field with two sidebands which are frequency-shifted by the frequency $\Omega$ of the modulation signal. Fig. 2.9 shows the phasor diagrams for this light field. Here we see that because of the additional sidebands, the resulting modulated field changes its phase relative to the carrier periodically over time.

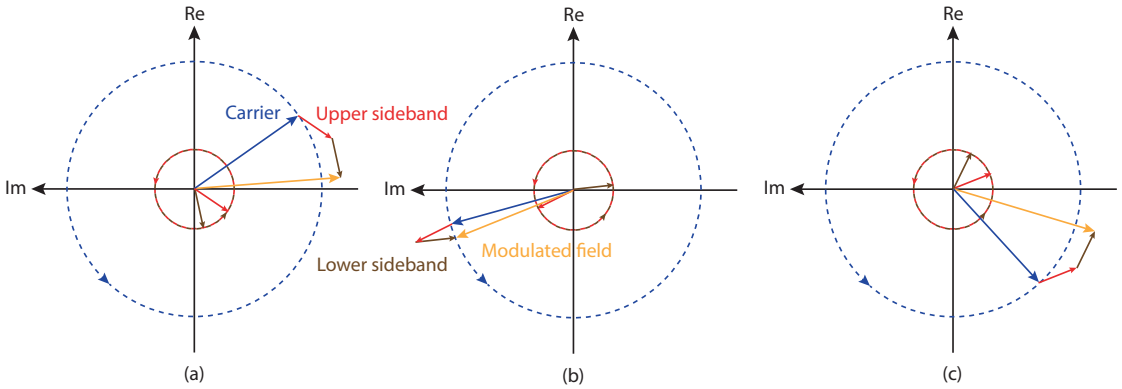


Figure 2.9: Phasor diagrams for phase modulated light. The carrier field is given by the blue vector rotating counterclockwise with the rate $\omega_0$. Red and green vectors show upper and lower sideband fields rotating counterclockwise with $\omega_0 \pm \Omega$. The sideband fields have a $\pi/2$ constant phase shift with respect to the carrier field (see equation 2.51) so that their sum is always orthogonal to the carrier. Thus, the resulting modulated oscillation vector (yellow vector as the sum of the other three) has a similar length compared to the carrier field but lags behind (a), nearly matches (b) or outruns (c) the carrier phase periodically with the modulation frequency $\Omega$.

If we now consider a phase modulation triggered by a gravitational wave, we can insert the phase change from equation 2.14 as the modulation signal from equation 2.43 and state the amplitude and phase of gravitational wave induced sidebands as

$$A_{\mathrm{sb}} = -\frac{w_0 h_0}{2 w_g} \sin\left(\frac{w_g L}{2c}\right) E_0 \tag{2.54}$$

and:

$$\varphi_{\mathrm{sb}} = \varphi_0 + \frac{\pi}{2} - \frac{\omega_0 L}{c} \pm \varphi_g \mp \frac{w_g L}{2c}. \tag{2.55}$$

As in equation 2.32, the term $\omega_0 L/c$ again results in zero, because $L$ is defined as a multiple of the wavelength so that this term is always a multiple of $2\pi$. This result will be used later for implementation of gravitational wave induced, phase modulated signal sidebands.

### 2.3.4 Readout Techniques

A typical simulation based on coupling matrices of optical components as described in section 2.3.2 solves the interferometer setup by calculating the light field amplitude and phase for every frequency and every output port. Using these fields one can then calculate error signals, frequency responses or sensitivity curves. Usually, different types of simulated detectors are required for these tasks. Here we define the light field on a detector placed at an interferometer output as

$$E = e^{\mathrm{i}\omega_0 t} \sum_{n=0}^{N} a_n e^{\mathrm{i}\omega_n t}, \tag{2.56}$$

and introduce different detectors which are usually used in steady-state interferometer simulation.

First, there is the *amplitude detector*, which is a pure hypothetical tool in simulation as only light intensity or power can be measured directly in experimental tests. It simply calculates the amplitudes of the light field at a specified frequency $\omega_m$, distinguishing between positive and negative frequencies:

$$z = \sum_n a_n \quad \text{with} \quad \{n \mid n \in \{0, \ldots, N\} \wedge \omega_n = \omega_m\}. \tag{2.57}$$

A second type of detector is the *photodetector* which measures the intensity of the light field as defined by

$$S_0 = |E|^2 = E \cdot E^* = \sum_{i=0}^{N} \sum_{j=0}^{N} a_i a_j^* e^{\mathrm{i}(\omega_i - \omega_j)t}. \tag{2.58}$$

By applying a low-pass filter, the output of frequency independent components ($\omega_i - \omega_j = 0$) can be calculated as a real number effectively using the photodetector to measure only the DC power:

$$S_{0,\mathrm{DC}} = \sum_{i=0}^{N}\sum_{j=0}^{N} a_i a_j^* \quad \text{with} \quad \{i,j \mid \omega_i = \omega_j\}\,. \tag{2.59}$$

Another detector type is the *photodetector with demodulation* to measure the power at a certain frequency. Here, the field is multiplied with a cosine with demodulation frequency $\omega_x$ and demodulation phase $\varphi_x$ which is called the *local oscillator*:

$$\begin{aligned} S_1 &= S_0 \cdot \cos\left(\omega_x t + \varphi_x\right) = S_0 \frac{1}{2}\left(e^{\mathrm{i}(\omega_x t+\varphi_x)} + e^{-\mathrm{i}(\omega_x t+\varphi_x)}\right) \\ &= \frac{1}{2}\sum_{i=0}^{N}\sum_{j=0}^{N} a_i a_j^* e^{\mathrm{i}(\omega_i-\omega_j)t} \cdot \left(e^{\mathrm{i}(\omega_x t+\varphi_x)} + e^{-\mathrm{i}(\omega_x t+\varphi_x)}\right). \end{aligned} \tag{2.60}$$

Then again a low pass filter is applied to measure only the DC power after multiplication with the local oscillator:

$$S_{1,\mathrm{DC}} = \sum_{i=0}^{N}\sum_{j=0}^{N} \frac{1}{2}\left(A_{ij} e^{-\mathrm{i}\varphi_x} + A_{ij}^* e^{\mathrm{i}\varphi_x}\right) \quad \text{with} \quad \{i,j \mid \omega_{ij} = \omega_x\} \tag{2.61}$$

where $A_{ij} = a_i a_j^*$ and $\omega_{ij} = \omega_i - \omega_j$. These three detector types allow us to measure the amplitude and power of the carrier field as well as of signal sidebands which is necessary to calculate the noise-to-signal ratio from equation 2.19 by taking into account the quantum noise described in the next two sections.

### 2.3.5 The Sideband Formulation of Quantum Noise

Section 2.2.1 gave a brief overview of different noise sources present in a gravitational wave detector. One of these was quantum noise which appears in two forms: shot noise and radiation pressure. In this section, we will follow Bond et al. [25] to sketch a derivation of the power spectral density $S_P$ which is needed for the calculation of the noise-to-signal-ratio from equation 2.19 and which we will use later to implement shot noise in our simulator.

The PSD of the noise in some photocurrent can be defined as the single-sided cross-power-spectral-density

$$S_I(\omega)\delta\left(\omega-\omega'\right)=2\left\langle I(\omega)I^*\left(\omega'\right)\right\rangle. \tag{2.62}$$

The photocurrent $I$ is proportional to the detected light power, so we can calculate it using

$$I(t)\sim P(t)=E(t)E^*(t). \tag{2.63}$$

To calculate $E(t)$, we first represent quantum fluctuations as noise in both amplitude and phase of a carrier field:

$$E(t)=\left[a_0+n_a(t)\right]e^{\mathrm{i}\omega_0 t+n_\phi(t)/a_0}+\text{c.c}=\left[a_0+n_a(t)+\mathrm{i}n_\phi(t)\right]e^{\mathrm{i}\omega_0 t}+\text{ c.c.} \tag{2.64}$$

where $n_a$ and $n_\phi$ are real amplitudes of phase and amplitude fluctuations which in the frequency domain can form the complex noise

$$q(\omega)=n_a(\omega)+\mathrm{i}n_\phi(\omega). \tag{2.65}$$

$n_a(\omega)$ and $n_\phi(\omega)$ are characterized by Gaussian probability density functions with mean $\mu_{a,\phi}=0$ and variance $\sigma^2_{a,\phi}$. Noise with equal and minimum variance $\sigma^2_{a,\phi}$ present in a light field can be used to represent vacuum noise which can be understood as the photon being incoherently created and annihilated at all frequencies. $q(\omega)$ is then a vacuum noise sideband representing an incoherent and non-deterministic signal. For a more rigorous approach than this semi-classical one, we refer to the review by Danilishin and Khalili [3] where the sidebands are replaced by actual quantum mechanical operators.

In their two-photon formalism, Caves and Schumaker [52, 53] have shown that amplitude and phase, the two quadratures of the light field, form an observable conjugate pair. Thus, they cannot be measured simultaneously without some uncertainty and their fluctuation variances follow the Heisenberg uncertainty principle

$$\sigma_\phi\sigma_a\geq\frac{\hbar\omega}{2}. \tag{2.66}$$

We can now consider a carrier field with a continuum of such vacuum noise sidebands relative to the carrier field frequency in the positive frequency spectrum:

$$E(t)=\frac{a_0}{2}e^{\mathrm{i}\omega_0 t}+\frac{e^{\mathrm{i}\omega_0 t}}{2}\int_{-\omega_0}^{\infty}q\left(\omega_0+\omega\right)e^{\mathrm{i}\omega t}d\omega+\text{ c.c. }. \tag{2.67}$$

Now we can constrain the range of the noise sidebands to the bandwidth $B \ll \omega_0$ of gravitational wave induced signals in which they will actually affect the sensitivity:

$$E(t) = \frac{1}{2}\left[a_0 + \int_{-B}^{B} q\left(\omega_0+\omega\right) e^{\mathrm{i}\omega t} d\omega\right] e^{\mathrm{i}\omega_0 t} + \text{ c.c.} \tag{2.68}$$

Inserting this into equation 2.63 and neglecting terms of the order $q^2$ yields

$$\begin{aligned} I(t) = |a_0|^2 &+ a_0^* \int_{-B}^{B} q\left(\omega_0+\omega\right) e^{\mathrm{i}\omega t} d\omega \\ &+ a_0 \int_{-B}^{B} q^*\left(\omega_0+\omega\right) e^{-\mathrm{i}\omega t} d\omega + O\left(q^2\right). \end{aligned} \tag{2.69}$$

Taking the Fourier transform and calculating the single-sided cross-power-spectral-density as in equation 2.62 for $0 < \omega \leq B$ reduces the calculation to one upper and one lower sideband:

$$S_I(\omega)\delta\left(\omega-\omega'\right) = 2P_0\left(\left\langle q_+ q_+'^*\right\rangle + \left\langle q_- q_-'^*\right\rangle\right) + 2a_0^2\left\langle q_- q_+'\right\rangle^* + 2a_0^{2*}\left\langle q_+ q_-'\right\rangle \tag{2.70}$$

where we used the notation $q\left(\omega_0 \pm \omega\right) \Rightarrow q_\pm$and $q\left(\omega_0 \pm \omega'\right) \Rightarrow q'_\pm$. For vacuum noise, the amplitude and phase fluctuations at different frequencies are independent, so for the covariance between to vacuum sidebands we find:

$$\begin{aligned} \left\langle q(\omega) q^*\left(\omega'\right)\right\rangle &= \frac{\hbar\omega}{2}\delta\left(\omega-\omega'\right) \\ \left\langle q(\omega) q\left(\omega'\right)\right\rangle &= 0. \end{aligned} \tag{2.71}$$

Inserting these covariance equations into equation 2.70 gives us the final noise PSD for a single carrier field with vacuum noise:

$$\begin{aligned} S_I\left(\omega_0 \pm \omega\right)\delta\left(\omega-\omega'\right) &= 2P_0\left(\left\langle q_+ q_+'^*\right\rangle + \left\langle q_- q_-'^*\right\rangle\right) \\ &= P_0\left(\hbar\left(\omega_0+\omega\right) + \hbar\left(\omega_0-\omega\right)\right)\delta\left(\omega-\omega'\right) \\ S_I\left(\omega_0 \pm \omega\right) &= 2P_0\hbar\omega_0. \end{aligned} \tag{2.72}$$

This means, that vacuum fluctuations result in a noise source that is only dependent on the power and frequency of the carrier field. In the two-photon formalism the usual formalism is in terms of field quadratures, that is the cosine quadrature $a_c(\omega)$ and the sine quadrature $a_s(\omega)$. These can be calculated from the sidebands $q_+$ and $q_-^*$ using

$$\begin{bmatrix} a_c \\ a_s \end{bmatrix} = A \begin{bmatrix} q_+ \\ q_-^* \end{bmatrix}, \quad A = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -\mathrm{i} & \mathrm{i} \end{bmatrix}. \tag{2.73}$$

To calculate the noise power spectral density (PSD) for a carrier field from pure vacuum noise, equation 2.72 is all that is needed. However, the vacuum noise has its origin in multiple sources within the optical setup: laser noise (from the spontaneous emission of photons in the gain medium [54, 55, 56]), optical losses at mirrors or beam splitters (due to the Fluctuation Dissipation Theorem [57, 58]) and squeezed vacuum field inputs. If this optical system contains non-linear behaviour such as squeezing or radiation pressure effects, the vacuum noise sidebands will get correlated and need to be propagated through the entire system, similar to signal sidebands. For large interferometer models, the number of noise sources is usually much greater than the number of detectors. Therefore, the interferometer simulator FINESSE implements the so-called multiple-input single-output (MISO) method [59]. Here, the transfer functions of all noise sources for a single output are computed together and combined to calculate the final noise PSD. Starting from the input output equation with the interferometer matrix $\mathbb{M}$, we find:

$$\begin{aligned} \mathbb{M}\ \vec{q}_{\mathrm{out}} &= \vec{q}_{\mathrm{in}} \\ \mathbb{M}\ \vec{q}_{\mathrm{out}}\ \vec{q}_{\mathrm{out}}^{\,\dagger}\ \mathbb{M}^{\dagger} &= \vec{q}_{\mathrm{in}}\ \vec{q}_{\mathrm{in}}^{\,\dagger} \\ \vec{q}_{\mathrm{out}}\ \vec{q}_{\mathrm{out}}^{\,\dagger} &= \mathbb{M}^{-1} \vec{q}_{\mathrm{in}}\ \vec{q}_{\mathrm{in}}^{\,\dagger}\ \mathbb{M}^{-1\dagger} \\ \left\langle \vec{q}_{\mathrm{out}}\ \vec{q}_{\mathrm{out}}^{\,\dagger} \right\rangle &= \mathbb{M}^{-1} \left\langle \vec{q}_{\mathrm{in}}\ \vec{q}_{\mathrm{in}}^{\,\dagger} \right\rangle \mathbb{M}^{-1\dagger} \\ \mathbb{V}_o &= \mathbb{M}^{-1} \mathbb{V}_i \mathbb{M}^{-1\dagger} \end{aligned} \tag{2.74}$$

To implement equation 2.72 with potential non-linear behavior from squeezers or optomechanics, we thus have to calculate the co-variance matrix $\mathbb{V}_o$ of the output noise sidebands. According to equation 2.74, this can be done efficiently through one inversion of the interferometer matrix and two matrix products. This results in the co-variance between any noise sidebands due to any noise sources at any output port in the optical system. Equation 2.72 then is fully implemented by

$$S_I = \frac{\vec{s}_c^{\,\dagger} \mathbb{V}_o \vec{s}_c}{2}, \tag{2.75}$$

where $\vec{s}_c$ is the solution of the carrier field at the position of the photodetector which acts as a selection vector and the factor $1/2$ comes from the demodulation of the photodiode output at the signal frequency [60].

For laser noise and surface losses, the diagonal of the input noise matrix $\mathbb{V}_i$ gets filled with vacuum noise, so depending on the chosen vacuum unit e.g. similar to equation

2.71 with $\hbar\omega/2$. As briefly mentioned in section 2.2.1, shot noise can be reduced by injecting squeezed light into the optical system. Squeezed light can be obtained as a result of parametric down conversion (PDC) in optically nonlinear crystals [61] or through a ponderomotive nonlinearity as induced by optomechanics described in section 2.3.6. For details about PDC and a formal derivation of the squeezing transformation, we refer to [62]. In general, a pump light emits photons with frequency $2\omega_0$ which give birth to correlated photons of frequencies $\omega_1$ and $\omega_2$ due to the nonlinear interaction with the crystal. This process can be described by the so-called PDC Hamiltonian. Solving the Heisenberg equations for this Hamiltonian in the interaction picture results in a squeezing transformation for vacuum noise sidebands that takes the form [53, 60]

$$s\left(\omega_0 \pm \omega\right) = \cosh(r) \cdot q\left(\omega_0 \pm \omega\right) + \sinh(r) \mathrm{e}^{2\mathrm{i}\phi} \cdot q^{\dagger}\left(\omega_0 \mp \omega\right), \tag{2.76}$$

where $s\left(\omega_0 \pm \omega\right)$ are the amplitudes of squeezed fields and $q\left(\omega_0 \pm \omega\right)$ are the amplitudes of the coherent sidebands. Thus, to include a squeezed vacuum source in the noise input matrix $\mathbb{V}_i$, both diagonal and off-diagonal entries must be filled according to equation 2.76 as the two sidebands are now correlated.

Reducing shot noise via squeezing can be interpreted as reducing the uncertainty in the phase quadrature of a light field which reduces the power fluctuations seen by the output port photodetector. However, if not doing frequency dependent squeezing, this will expand the uncertainty in the amplitude quadrature which in turn increases the radiation pressure noise due to optomechanics described in the next section.

### 2.3.6 Simulating Optomechanics

Besides shot noise, the other quantum noise type is radiation pressure. In this section we will give a short introduction to a simplified version of this effect by following the review by Bond et al. [45] and the FINESSE manuals [9, 8].

According to Maxwell's theory of light, an optical field propagating in a vacuum exerts a force on a surface that is proportional to its time-averaged Poynting vector. In the frequency spectrum, this force can be written as [63]

$$F(\omega) = \frac{P(\omega)\cos\alpha}{c} \tag{2.77}$$

where $\alpha$ is the angle of incidence, $c$ the speed of light and $P(\omega)$ the power fluctuation at frequency $\omega$. With the high circulating power in the arm cavities of gravitational wave detectors, this force is often large enough to displace the cavity mirrors by multiple wavelengths of the light. This relatively large displacement is a control problem that is

addressed by the process of lock acquisition [64, 65]. However, even in a well controlled interferometer state, there are steady state optomechanical effects causing surface motion as the result of perturbations in the optical field. These radiation pressure effects can have significant effects on the sensitivity, e.g. through parametric instabilities [66] or ponderomotive squeezing through optical springs [67, 68, 69].

The general process of such effects is that disturbances in light power induce some motion of a surface which in turn creates phase modulation sidebands around any carrier that is reflected from it. Considering e.g. a mirror with four ports, the applied force can be written as

$$F(\omega) = \frac{P_{2i}(\omega) + P_{2o}(\omega) - P_{1i}(\omega) - P_{1o}(\omega)}{c}, \tag{2.78}$$

where optical fields coming from different sides also have opposing momentum reflected by the choice of sign. $P_{2i}$ and $P_{2o}$ stand for optical fields at input and output of one surface side, while $P_{1i}$ and $P_{1o}$ represent the optical input and output field from the other side. It is important to note that the calculation of the power terms for a carrier field with sidebands is given by

$$P(\omega) = q_+ E_0^* + q_-^* E_0, \tag{2.79}$$

including both upper and lower sidebands which get correlated in case of e.g. quantum noise sidebands. The induced motion of a mirror due to $N_f$ separate forces can be described by

$$\delta z(\omega) = H(\omega) \sum_{n=0}^{N_f} F_n(\omega), \tag{2.80}$$

where $H(\omega)$ is the *mechanical susceptibility* or *mechanical transfer function* from an applied force to the longitudinal mirror motion. If we assume a free-floating mirror (e.g. due to a suspension system) we have

$$H(\omega) = -\frac{1}{m\omega^2}, \tag{2.81}$$

which is a direct result of taking the Fourier transform of Newton's second law. What is now missing to close the loop is the transition from this mirror motion to an optical phase change. As described in equation 2.31, a change in length is accompanied by a phase change. In this case, the induced phase difference will be

$$\varphi = -k\delta z(\omega) \tag{2.82}$$

and the result is a phase modulation and the creation of an additional pair of sidebands as described in section 2.3.3. The formalism described in this section is a simplified version of optomechanics which in general would not be possible to simulate using the linear system approximation from section 2.3.2. Therefore, interferometer simulators are usually based on reasonable approximations which simplify the calculation of optomechanical effects under certain conditions. Specifically, due to equation 2.81 we can assume that high-frequency power fluctuations are negligible. Low-frequency power fluctuations are assumed to be very small, so that there is a clear separation between the carrier field and any created sidebands. Third, we assume that the resulting mirror motion of any surface is small with respect to the wavelength, that is we assume $|\delta z(\omega)| \ll \lambda$. In the case of a steady state gravitational wave detector all of these assumptions are valid and used by popular interferometer simulation software as described in the next section.

### 2.3.7 Interferometer Simulation Software

The central goal of interferometer simulators is to study important physical features of interferometer setups in order to find possible improvements or minimize the probability of unforeseen problems in the commissioning phase. The first prototype codes for this purpose appeared in 1988, written by Jean Yves-Vinet [70] and were later used to gradually implement ever more complex simulation tools for optical propagation, control system feedback and mechanical suspensions [71, 72].

The design of complex interferometers usually makes use of different kinds of simulation tools [73]. Firstly, full system time domain simulators address non-stationary and nonlinear processes such as e.g. the lock acquisition process. These simulators are computationally expensive and are complex in structure. Two popular full system time domain tools developed at LIGO and VIRGO are E2E [71] and SIESTA [72]. E2E was used for the design and continuous improvement of the lock acquisition design during the initial LIGO commissioning phase as well as for improvements in alignment control and first sensitivity curve calculations of initial LIGO [74, 75, 76]. These simulators also include suspension simulations for quadruple and triple pendulums [77] as well as simulations of seismic isolation systems [78].

A second class of simulation tools are programs which calculate the details of field profiles in the interferometer using details of optics like e.g. the mirror surface aberration [79, 70]. Ideally, the full system time domain simulators would natively include these high detail field profiles. However, due to computational complexity the time domain models mostly approximate fields using modal models with limited number of modes while the

high detail profiles are studied separately using a static model based on the fast Fourier transform (FFT) method. These simulation methods have been used to decide the optimal choice of mirror curvature, to improve the thermal compensation system and to study the effect of beam splitter curvature [80]. An example of such a high-detail field profile simulator is a tool called FFT [80, 79].

The third group of simulators are simulators which describe only stationary and linear systems [59]. They are not capable of simulating time-dependent processes such as e.g. the lock acquisition process or the time evolution of the light fields in cavities. After an interferometer acquires lock and becomes stationary, it can be treated as a single linear system and these steady-state simulators become applicable. Computationally they are less demanding and easier to understand as a user because many non-linear and transient behaviors are assumed away. Examples of popular FFT-based steady-state simulators are OSCAR [81, 82] and SIS [83]. Popular frequency domain simulators that are in use today are Optickle [84], MIST [85] and FINESSE [7, 8, 9]. These simulators have been deployed for design and commissioning tasks for LIGO, GEO600 and third generation detectors such as the Einstein Telescope [59, 86] as well as for simulating optomechanical effects [87] and running optimizations on detector designs [88, 6]. The different frequency domain simulators differ e.g. in terms of how they compute higher order modes and which interferometer matrix schemes they are using [85].

In addition to being applied in practice for commissioning, design and optimization tasks, there also exist studies comparing the results from different simulators like FINESSE, Optickle and OSCAR [89, 90]. While most results match between simulators, there are also unexplained differences where it is not clear which simulator produces more accurate values as reference results are missing.

We use the open source FINESSE simulator [7, 9] as a template for our new Differometor simulator. The idea for FINESSE first came up in 1997 when Andreas Freise and Gerhard Heinzel were working on a 30 m prototype interferometer with Dual Recycling in Garching. Since then, Andreas Freise developed FINESSE during his work at GEO 600 until Daniel Brown took over development around 2013 and later developed the open-source FINESSE 3 version that is the template for Differometor [9, 59].

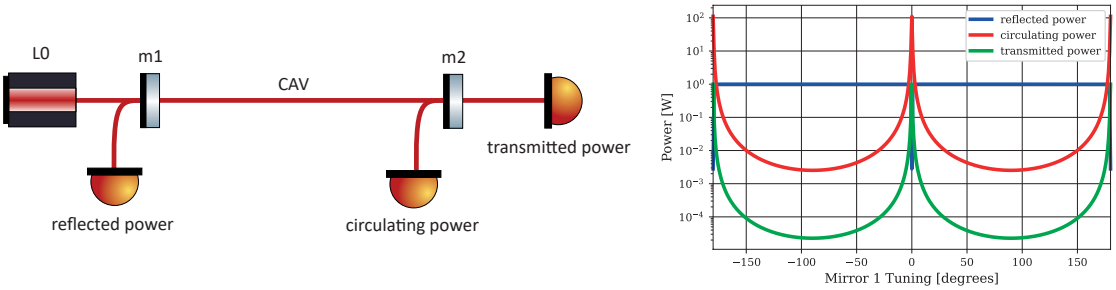


Figure 2.10: Left: cavity setup as described in code block 2.1. Right: power outputs from the three shown detectors, simulated with FINESSE.

FINESSE implements all mechanisms described in this chapter. It describes optical components via coupling matrices, offers different methods for light field modulation and readout techniques, models quantum noise in the sideband formulation and simulates different optomechanical effects. It can perform analysis using plane waves or Hermite-Gaussian modes where the latter allows for the computation of mode matching and misalignment effects. FINESSE comes with its own model syntax which allows the definition of complex optical systems. For example, a cavity as shown in Fig. 2.10 can be defined and simulated as shown in code block 2.1 where the simulator sweeps over possible tuning values and detects the power output at different points within the system.

```python
import finesse
finesse.configure(plotting=True)

kat = finesse.Model()
kat.parse(
    """
    # Add a Laser named L0 with a power of 1 W.
    l L0 P=1

    # Space attaching L0 <-> m1 with length of 0 m (default).
    s s0 L0.p1 m1.p1

    # Highly reflective input mirror of cavity
    m m1 R=0.99 T=0.01

    # Intra-cavity space with length of 1 m.
    s CAV m1.p2 m2.p1 L=1

    # Highly reflective end mirror of cavity.
    m m2 R=0.991 T=0.009

    # Power detectors on reflection, circulation and transmission.
    pd reflected_power m1.p1.o
    pd circulating_power m2.p1.i
    pd transmitted_power m2.p2.o
    """
)

out = kat.run("xaxis(m1.phi, lin, -180, 180, 400)")
out.plot(logy=True)
```

Code 2.1: Defining and simulating a cavity setup in Finesse. This example was taken from the official FINESSE 3 documentation [9]. Setup and power outputs are shown in Fig. 2.10.

## 2.4 Digital Discovery

Due to the availability of large datasets, improvements in parallel computing and storage hardware and the invention of new algorithms, the power of artificial intelligence (AI) methods has vastly increased since the early 2010s. In recent years, AI methods are also being increasingly integrated into scientific discovery, a process that is sometimes called *Digital Discovery* and often has the ultimate goal of leveraging these AI techniques to gain new scientific understanding [91]. In this section, we give an overview of recent

uses of AI methods for science based on the reviews by Krenn et al. [91] and Wang et al. [92] and then concentrate on one specific approach to digital discovery in the field of gravitational wave detectors which will be relevant for the optimizations in later chapters of this thesis. Finally, we explain the term *Differentiable Programming* as the main programming paradigm for this thesis.

### 2.4.1 AI for Scientific Understanding

Krenn et al. [91] identify three fundamental dimensions of how AI systems can contribute to new scientific understanding. As *computational microscopes* they can help humans to inspect and simulate complex physical systems, revealing insights which humans might then turn into new understanding. As a *resource of inspiration*, AI methods can predict and select promising scientific hypotheses or find surprising inconsistencies between data and theoretical predictions which might again lead humans to gain new scientific understanding. Finally, as *agents of understanding*, future AI models might be able to gain such understanding themselves and then pass it on to humans.

One process on the path towards scientific understanding is the formulation of meaningful, testable hypotheses. AI methods can be used to support this process by predicting potential hypothesis candidates. They can for example prioritize molecules for experimental investigation in drug discovery [93], screen material candidates for technical applications [94, 95] or predict the 3D atom coordinates of proteins from amino acid sequences [96]. Reinforcement-learning algorithms and evolutionary algorithms can be used to navigate combinatorial, discrete hypothesis spaces, like making discrete decisions in the design process of molecules with specific pharmaceutical properties [97, 98], while variational auto-encoders can map such discrete search spaces into differentiable ones, which allow the application of gradient-based methods [99, 100].

AI methods can then also aid the evaluation of selected hypotheses. For example, reinforcement-learning approaches can be used for experiment planning [101, 102] and control tasks [103, 104]. So-called neural solvers combine physics and deep learning by integrating domain knowledge into neural networks [105, 106] which makes it easy to evaluate selected hypothesis in the form of differential equations. Simulation-based inference [107] allows for efficient hypothesis testing by comparing simulated data with observed data, enabling the estimation of model parameters and uncertainties.

Our ultimate goal with Differometor is to use AI to navigate a search space of scientific hypotheses. In our case, these hypotheses are possible designs for new gravitational wave detectors. A proof-of-concept of this design approach has been recently demonstrated by Krenn et al. [6] and will be further discussed in the next section. Other recent works have demonstrated successful automated hardware design in the field of quantum optics

[5, 4], quantum circuits [108, 109, 110], photonic structure [111, 112] and microscopy [113].

### 2.4.2 Digital Discovery of Gravitational Wave Detectors

For the digital discovery of new gravitational wave detectors, Krenn et al. [6] have introduced *Urania*, a parallelized hybrid local-global optimizer. They start by defining the search space of experimental detector topologies as a combinatorial problem. With one laser, four optical elements and two detectors, there are already 3000 unique topologies that could be built. Increasing the number of elements to 10 and assuming that each element can be described by two continuous properties already yields 100 million unique 20-dimensional search spaces. They argue that instead of exploring this search space via rational design, that is letting experts come up with novel ideas, one can phrase this as an exploration task well suited for computational approaches.

They then transform this discrete, combinatorial search space into a continuous search space suited for gradient-based optimization methods. To do this, they invent a quasi-universal interferometer (UIFO) which avoids discrete component choices by including enough components in a highly expressive topology which can by itself encode a large portion of setups in the search space.

Fig. 2.11 A) shows an example of such a setup, consisting of $3 \times 3$ beam splitter or Faraday isolator cells surrounded by lasers, squeezers and detectors. By choosing the right parameters it is then possible to encode different topological structures like for example the simplified aLIGO setup from Fig. 2.3 as shown in Fig. 2.11 B). A UIFO of size $3 \times 3$ has up to 187 parameters. UIFOs of size $4 \times 4$ have 310 parameters, while UIFOs of size $5 \times 5$ reach up to 463 parameters.

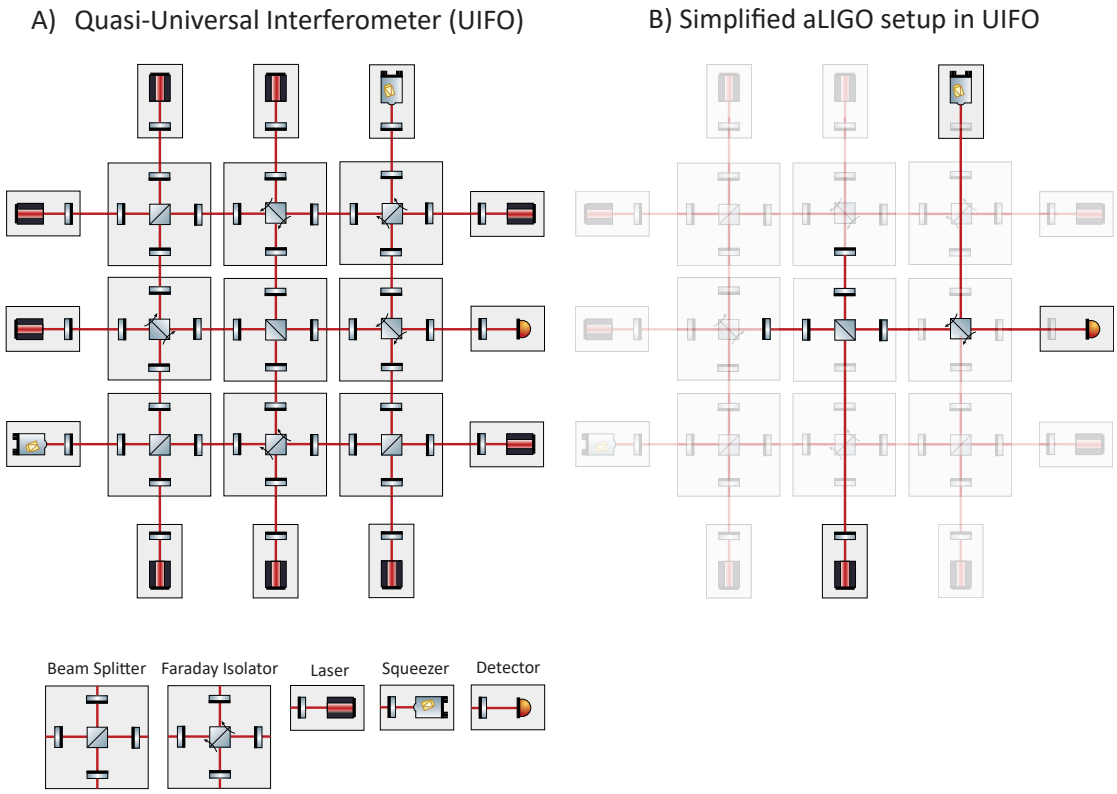


Figure 2.11: **A)** The quasi-universal interferometer (UIFO) template from Krenn et al. [6] for the size $3 \times 3$. It consists of nine cells with either beam splitter or Faraday isolator with four enclosing mirrors. Light from lasers or squeezers is injected into the setup from the boundaries. A detector is placed at one of the boundaries. **B)** The simplified aLIGO setup from Fig. 2.3 encoded in the UIFO from A). Transparent components are not used.

They estimate the quality of a setup by calculating its sensitivity (see section equation 2.19) with the interferometer simulator FINESSE described in section 2.3.7 and by adding some parameter constraints and limits for the power throughput. The loss function they use is given by

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{Strain}} + \alpha \cdot \text{Penalty}_{\text{hard}} + \beta \cdot \text{Penalty}_{\text{soft}} + \gamma \cdot \text{Penalty}_{\text{bleach}} \tag{2.83}$$

where $\alpha, \beta$ and $\gamma$ are hyperparameters and

$$
\begin{aligned}
\mathcal{L}_{\text{Strain}} &= \int_{f_0}^{f_1} \log(S(f))df, \\
\text{Penalty}_{\text{hard}} &= \sum_{RO} f\left(p(RO), co_h, c_1\right), \\
\text{Penalty}_{\text{soft}} &= \sum_{TO} f\left(p(TO), co_s, c_2\right), \\
\text{Penalty}_{\text{bleach}} &= \sum_{\text{Det}} f\left(p(Det), co_d, c_3\right).
\end{aligned}
\tag{2.84}
$$

Here, $S(f)$ is the sensitivity at frequency value $f$, $p()$ stands for the power throughput at a transmitting object (TO) or reflecting object (RO) or at a detector (Det). $f()$ is a penalty function (e.g. a cumulative logistic function CDF) which is shifted by the power cutoff value $co$ and scaled by some hyperparameter $c$. These penalty functions are designed to protect interferometer components from too high power throughput and therefore from getting burned and damaged.

Their optimization algorithm Urania then starts with a pool of more than 1000 UIFO setups either randomly initialized or adapted from existing solutions for different frequency targets. Urania randomly chooses setups from this pool (weighting better performing ones higher) and either simplifies them or optimizes their parameters using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [114, 115, 116, 117]. Using this approach Krenn et al. find 50 UIFO configurations that outperform an optimized aLIGO baseline and show interesting detector designs.

At the core of this kind of digital discovery approach lies the simulator, which translates experimental designs to physical outputs used in objective functions. In case of the Urania algorithm, the simulator FINESSE represents the main bottleneck. Being implemented in NumPy [10], it can only run on CPU, so to identify the 50 UIFO configurations, Krenn et al. have spent around 1.5 million CPU-hours. The next section describes potential performance improvement techniques and introduces the main programming paradigm used for the implementation of our Differometor.

### 2.4.3 Differentiable Programming

Three potential techniques to speed up simulators are Just-In-Time (JIT) compilation, Graphics Processing Unit (GPU) execution and auto-differentiation.

JIT compilation [118] dynamically compiles code portions during runtime, optimizing its behavior based on runtime requirements such as the used hardware and thereby increasing computational efficiency. GPU execution allows multiple calculations to be performed concurrently through specialized hardware units and thereby speeds up simulators that

allow for such execution. Auto-differentiation [119, 120] augments standard code execution with the automatic and efficient calculation of its derivative. It implicitly redefines the semantics of operators to propagate derivatives via the chain rule of differential calculus. While numerical gradient approximations would need to call the simulator multiple times per variable to determine its gradient, a simulator that implements automatic differentiation, yields the gradient of its output with only one function evaluation.

A prominent programming framework that combines these three features is JAX [11], which is developed by Google Research and was originally conceived for complex deep learning tasks. JAX is designed to follow the workflow and structure of NumPy as closely as possible and therefore allows for easy integration into existing Python code. We use this framework in later chapters to implement our differentiable interferometer simulator. Due to the mentioned features, JAX is well-suited for scientific programs and simulators. However, offering these features comes with its own set of constraints. First, JAX features like e.g. JIT compilation only work for Python functions that are functionally pure, i.e. functions where all inputs are passed as parameters and all results are output through the function's return statement. Second, JIT compilation only works when function outputs have static shapes, i.e. the shapes of output arrays are not dependent on the values of function inputs. Third, Python control flow and logical operators like *if* and *else* are executed at JIT compilation time and can't depend on the values of function inputs either. For example, the Python functions from code block 2.2 cannot be JIT compiled.

```
@jit
def f(x):
  if x < 3:
    return 3. * x ** 2
  else:
    return -4 * x

@jit
def nansum(x):
  mask = jnp.invert(jnp.isnan(x))  # boolean mask selecting non-nan values
  x_without_nans = x[mask]
  return x_without_nans.sum()
```

Code 2.2: Examples of Python functions that throw errors during JIT compilation. Examples taken from [11].

In general, auto-differentiation is the workhorse of *Differentiable Programming* [121], a generalization of neural networks and deep learning and a programming paradigm

that enables the implementation of parameterized software modules that can be trained with gradient-based optimization. Differentiable Programming allows easy integration of deep learning methods into arbitrary algorithmic structures that go beyond matrix multiplications and nonlinear element-wise operations and is therefore well suited for the implementation of physical simulations. While deep learning frameworks like TensorFlow or PyTorch have dominated the Differentiable Programming landscape during the past decade, more recent general purpose frameworks like JAX, DiffTaichi [122] and Julia [123] are currently on the rise.

Recent works in the field of Differentiable Programming include differentiable pipelines for learning protein structure [124], rigid body simulations [125], complex computational fluid dynamics [126], molecular dynamics [127], matrix elements of high energy scattering processes [128] and hybrid classical-quantum systems [129].

# Chapter 3

# Differometor - A Differentiable Interferometer Simulator

In this chapter, we introduce Differometor, a new, differentiable steady-state frequency domain interferometer simulator implemented in Python using the JAX framework. Differometor is based on FINESSE 3, the open-source interferometer simulation program described in section 2.3.7. Differometor is designed to overcome the simulator bottleneck for the digital discovery of gravitational wave detectors (see section 2.4.2), but can also be used for other interferometer optimization or design tasks.

Building on the provided background in section 2.3, we first describe how the different parts of Differometor are implemented and demonstrate their functionality in simple examples (section 3.1 to 3.3). In section 3.4, we then verify Differometor against FINESSE 3, demonstrating the close agreement of the two simulators for different setups. In addition, we compare Differometor and FINESSE in terms of performance and demonstrate the achieved speedup based on just-in-time compilation and GPU support.

## 3.1 Setup Definition and Components

Similar to FINESSE, Differometor offers an easy way to define an optical setup. Instead of offering a custom syntax like FINESSE does, Differometor uses the Python package NetworkX [130] to define an optical setup as a graph structure. Nodes are optical components like lasers, mirrors or beam splitters and edges are spaces between these components. Fig. 3.1 shows an overview of available components and their default properties.

| Symbol | Component | Parameters |
|---|---|---|
|  | laser: | **properties** = { power: 1. , phase: 0. }, **target** = None, **port** = left, **direction** = in |
|  | squeezer: | **properties** = { db: 0. , angle: 0.}, **target** = None, **port** = left, **direction** = in |
|  | detector: | **properties** = { }, **target** = None, **port** = left, **direction** = in |
|  | qnoised: | **properties** = { }, **target** = None, **port** = left, **direction** = in |
|  | space: | **properties** = { length: 0. , refractive_index: 1. }, **source_port**: right, **target_port**: left |
|  | mirror: | **properties** = { reflectivity: 0.5 , loss: 0. , tuning: 0. } |
|  | beamsplitter: | **properties** = { reflectivity: 0.5 , loss: 0. , tuning: 0. , alpha: 45.} |
|  | directional_ beamsplitter: | **properties** = { } |
| | nothing: | **properties** = { } |
|  | free_mass: | **properties** = { mass: 40. }, **target** = None |
| **f** | frequency: | **properties** = { frequency: 1. } |
|  | signal: | **properties** = { amplitude: 1. , phase: 0. }, **target** = None |

Figure 3.1: A list of Differometor components and their parameters.

The laser has a phase parameter and a power parameter which translates into the light field amplitude $E_0$ from equation 2.21 using

$$P = \frac{\epsilon_0 c}{2} E_0 E_0^*, \tag{3.1}$$

where $\epsilon_0$ is the electric permeability of vacuum and $c$ the speed of light. Additionally, the laser has **target**, **port** and **direction** parameters with which it can get directly connected to a certain port of a component without using a space in between.

The squeezer has a db property which determines the squeezing magnitude in dB and translates into the squeezing parameter $r$ using

$$r_{db} = 20r \log_{10}(e). \tag{3.2}$$

The angle property determines the squeezing angle $\phi$. If $\phi = 0, 180$ the vacuum is purely amplitude squeezed while it is phase squeezed if $\phi = \pm 90$. Similar to a laser the squeezer has additional parameters to connect it directly to a component port without a space in between.

Due to the interferometer matrix implementation, Differometor always calculates all lights fields at all ports of the optical system. The detector component simply indicates at which port the light field should get measured and can be directly connected to a specific port using the **target**, **port** and **direction** parameters. The light field at the specified port can then be measured using different detector functions which implement the equations from section 2.3.4. First, the amplitude detector function returns the amplitude of the light field and is implemented as:

$$f_{\text{amplitude}}(z) = \sqrt{\frac{\epsilon_0 c}{2}}|z|. \tag{3.3}$$

Here, $z$ is the complex light field at the specified port. The power can be calculated via the power detector function:

$$f_{\text{power}}(z) = \frac{\epsilon_0 c}{2}|z|^2. \tag{3.4}$$

The phase detector function extracts the phase of the complex light field:

$$f_{\text{phase}}(z) = \arctan(\frac{b}{a}) \tag{3.5}$$

where b and a are the complex and real part of the light field $z = a + ib$. For signal demodulation, we use the demodulation detector function:

$$f_{\text{demodulation}}(c, s) = c^* s + c s^* \tag{3.6}$$

where $c$ is the complex carrier field at the port acting as local oscillator and $s$ is the complex signal field. The sensitivity detector function enables the calculation of sensitivity as shown in equation 2.19 and is implemented via

$$f_{\text{sensitivity}}(p, n) = \frac{n}{|p|} \tag{3.7}$$

where $n$ is the quantum noise calculated for this port and $p$ is the complex signal power at the port. The quantum noise detector from Fig. 3.1 indicates for which port the quantum noise should get calculated and sets the entries of the selection vector in equation 2.75.

Mirrors and beam splitters take a loss $L$ and a reflectivity fraction $R_f$ as input (called reflectivity in Fig. 3.1). Reflection and transmission coefficients can then be calculated via $R = R_f \cdot (1-L)$ and $T = (1-R_f) \cdot (1-L)$ where $R = r^2$ and $T = t^2$ and $R+T+L = 1$. The reason for the reflectivity fraction instead of the reflection coefficient itself is to be able to optimize both reflectivity and loss in an unconstrained way. If we would have $L$ and $R$ as input parameters, then it would lead to a constrained optimization problem which can be difficult to handle. With the reflectivity fraction, we can optimize both $R_f$ and $L$ independent from each other between 0 and 1.

The directional beam splitter from Fig. 3.1 acts like a Faraday isolator where light fields from the left port get propagated to the right port, light fields at the top get propagated to the left, light fields from the right get propagated to the bottom and light fields from the bottom get propagated to the top.

Free mass components can be attached to mirrors and beam splitters via their **target** parameter and will lead to optomechanical effects. Signal components induce space modulation and can be attached to spaces via their **target** parameter, by combining the component names which are connected by the space using _.

A component port always has one input and one output direction. Mirrors have two ports *left* and *right*, whereas beam splitters and directional beam splitters have four ports (*left*, *right*, *top*, *bottom*). When connecting components through spaces or attaching lasers or detectors, we can specify the port and the direction. As an example, we define the cavity setup similar to the FINESSE one from code block 2.1 as follows:

```python
import networkx as nx

G = nx.DiGraph()
G.add_node("l0", component="laser")
G.add_node("m1", component="mirror", properties={"reflectivity": 0.99})
G.add_node("m2", component="mirror", properties={"reflectivity": 0.991})

G.add_edge("l0", "m1")
G.add_edge("m1", "m2", properties={"length": 1})

G.add_node("reflected_power", component="detector", target="m1",
↪  direction="out")
G.add_node("circulating_power", component="detector", target="m2")
G.add_node("transmitted_power", component="detector", target="m2",
↪  port="right", direction="out")
```

Code 3.1: The definition of a simple cavity setup in Differometor.

Here, we add a laser with name *l0* together with two mirrors named *m1* and *m2* and connect all of these components through spaces with certain lengths. Then we add three detectors as shown in Fig. 2.10.

## 3.2 The Build Step

Given a defined setup, we now aim to implement a differentiable simulator which is designed for the digital discovery of gravitational wave detectors as described in section 2.4.2. This goal comes with a set of its own requirements and constraints.

First, in order to overcome the simulator bottleneck of existing approaches to gravitational wave detector optimization, one requirement for Differometor is to be performant in terms of execution and gradient calculation. As explained in section 2.4.3, speedup is possible through e.g. JIT compilation, GPU execution and auto-differentiation. All these features are offered by JAX, but entail additional programming constraints like the restriction to pure functions and static inputs (see section 2.4.3).

A second requirement comes from the optimization process. We would like to use Differometor to e.g. start with a randomly initialized interferometer setup and then optimize the parameters of the optical components in such a way, that the sensitivity of the setup is maximized. A suitable loss function to do this is equation 2.83. Thus, one requirement for the implementation of Differometor is the possibility to select certain parameters as targets for optimization.

Furthermore, the loss function from equation 2.83 requires the setup to run multiple times for different parameter values (e.g. for 100 different frequencies to approximate the sensitivity integral). Thus, a third requirement is the possibility to define a set of parameters which change during the simulation within corresponding value ranges. It should for example be possible to define a loss function that calls the simulator 100 times, each time with a different frequency.

The individual penalty terms of the loss function from equation 2.84 introduce a fourth requirement. In order to sum over the penalty functions for high power throughput, we need to know the light fields at all components within the optical system. In order to have access to all these fields, Differometor is implemented following the interferometer matrix approach explained in section 2.3.2. By solving the system of equations defined by the interferometer matrix, all light fields are calculated and can subsequently be used to compute power throughput penalties.

The first step in bringing all these requirements and constraints together is the realization that at its core, the simulation is simply about updating specific entries of the interferometer matrix and solving the resulting system of equations. The physics comes in through the structure of the matrix and through a small set of functions which are used to calculate the new matrix entries. This matrix updating has to be implemented in JAX using pure functions and static inputs only.

A Differometor simulation is therefore divided into 2 steps, the *build step* and the *simulation step*. The build step takes an optical setup defined as a graph structure and compiles it into a set of arrays which are needed to perform updates of the interferometer matrix and passed to the simulation step. The build step needs to be executed only once per optical setup and is implemented using NumPy [10] as it does not need to be differentiable.

Fig. 3.2 shows a schematic overview of the simulation of a cavity as defined in code block 3.1. The arrays which need to get compiled during the build step are marked in blue. First, there is the parameter array which contains the properties of all components of the optical system. When starting a simulation the user can specify parameters that should get optimized based on the simulation outcome. Based on this user input, the build step compiles arrays which hold these optimized values and the indices of where to place them in the parameter array (step 1 in Fig. 3.2).

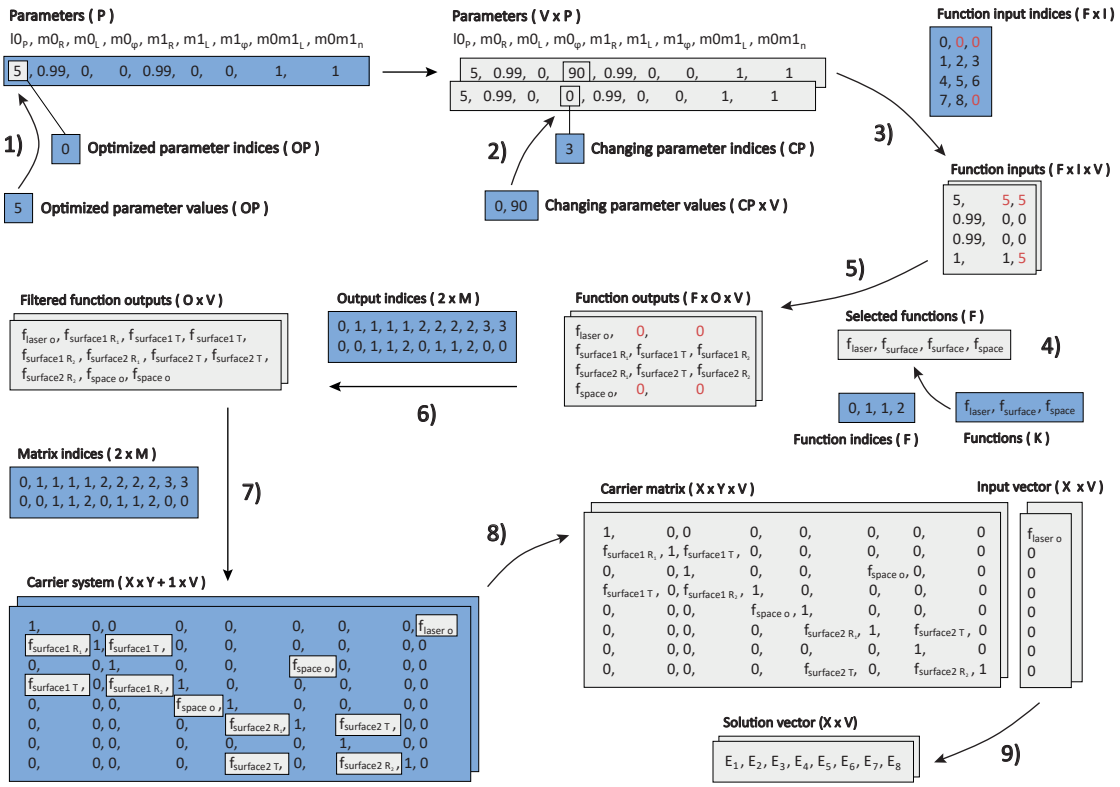


Figure 3.2: Differometor's carrier solving system. Blue arrays are outputs of the build step while gray arrays indicate newly composed arrays or values. 1) The parameter array gets updated with the values of the optimized parameters via an index array. 2) The parameter array gets expanded into a second dimension by the values of the changing parameters via another index array. 3) A third index array is used to build an array of function inputs necessary for the optical system at hand. The function input array contains dummy values to conform to the matrix shape. Dummy values are indicated in red. 4) Another index array is used to select the functions necessary for the optical system at hand. 5) The function input array is processed by the selected functions which results in a function output array. Again, the output array contains dummy values to conform to the matrix shape. 6) A fifth index array selects function outputs that should actually get placed in the carrier system. 7) Another index array delegates the selected function outputs to their positions in the carrier system. 8) The carrier system contains both carrier matrix and input vector and gets split to perform the final solving step. 9) The carrier system gets solved and outputs the solution vector containing the fields at all input and output ports of the optical system.

Additionally, the user can also specify parameters which change during simulation. This initiates multiple parallel simulations for which the build step compiles arrays holding the values of the changing parameters as well as the indices of where to place them in the parameter array (step 2 in Fig. 3.2).

To calculate the entries of the coupling matrices of different components, the build step also compiles function and index arrays encoding which parameters should get processed by which functions and how the outputs of these functions should then get distributed over the interferometer matrix. Specifically, the build step compiles an index array which dependent on the optical system at hand, selects the functions necessary to compute the updated interferometer matrix entries from a static list of all available functions (step 4 in Fig. 3.2). The build step also compiles an index array indicating which of the parameters are needed as inputs to these functions (step 3 in Fig. 3.2). To make use of the JAX feature *vmap* offering automatic vectorization, the function input index array contains dummy elements to be able to place all function inputs in one array even so some functions might need a different number of inputs. For the same reason, the output array of the function application also contains dummy elements as some functions might return more values than others. To filter out potential dummy elements, the build step compiles another output index array filtering only actual return values (step 6 in Fig. 3.2). To be able to distribute these return values over the interferometer matrix the build step compiles another matrix index array encoding the positions of all optical elements and their coupling matrix structures (step 7 in Fig. 3.2). The functions used to compute the updated interferometer matrix entries are the core implementation of the physics described in chapter 2 and are explained in the next few sections.

## 3.3 The Simulation Step

The simulation step updates and solves the system of linear equations defined by the interferometer matrix of the optical system based on the information it gets from the build step. To compute the updated matrix entries it makes use of different functions that implement the physics described in chapter 2. We can categorize these functions into the carrier system, the signal system, the quantum noise system and the optomechanics system which will be described in the next sections.

### 3.3.1 The Carrier System

The *carrier system* handles light fields without a frequency offset from the default frequency. Carrier light fields are usually subject to some kind of modulation, e.g. by a modulator or by a signal as described in section 2.3.3. The core of the carrier system is the *carrier matrix*, an interferometer matrix that describes the entire optical setup. As described in section 2.3.2 this matrix is built from coupling matrices of the different components of the optical system along the matrix diagonal connected by space entries in the off-diagonal parts.

There are five core functions necessary to simulate the carrier system. The first function is the laser function with power $P$ and phase $\varphi$ (in radians) as input arguments:

$$f_{\text{laser}}(P, \varphi) = \sqrt{\frac{2P}{\epsilon_0 c}} \cdot \exp(i\varphi). \tag{3.8}$$

This is equation 2.21 with the power scaling factor introduced in equation 3.1.

The second function is the space function with frequency offset $\Delta f$, length $L$ and refractive index $n$ as input arguments:

$$f_{\text{space}}(\Delta f, L, n) = -\exp(-\mathrm{i}2\pi \frac{\Delta f}{c} nL). \tag{3.9}$$

This calculates the space connector entries as described in equation 2.41. The next three functions can be used for both mirrors and beam splitters to calculate all entries of their coupling matrices described by equations 2.37 and 2.40. Their inputs are reflectivity $R$, tuning $\phi$, frequency offset $\Delta f$, default frequency $f_0$, left refractive index $n_1$, right refractive index $n_2$, and tilt angle $\alpha$. The three equations are derived from the equations 2.38 and 2.36 and given by

$$f_{\text{aux1}}(n_1, n_2, \alpha) = \cos\left(\arcsin\left(\frac{n_1}{n_2}\sin\alpha\right)\right) \tag{3.10}$$

$$f_{\text{aux2}}(\phi, \Delta f, f_0, n_1, \alpha) = 2\phi n_1 \cos\alpha \left(1 - \frac{\Delta f}{f_0}\right) \tag{3.11}$$

$$f_{\text{aux3}}(\phi, \Delta f, f_0, n_1, n_2, \alpha) = 2\phi n_2 f_{\text{aux1}} \left(1 - \frac{\Delta f}{f_0}\right) \tag{3.12}$$

$$f_{\text{refl1}}(R, \phi, \Delta f, f_0, n_1, \alpha) = -\sqrt{R}\exp\left(i f_{\text{aux2}}\right) \tag{3.13}$$

$$f_{\text{refl2}}(R, \phi, \Delta f, f_0, n_1, n_2, \alpha) = -\sqrt{R}\exp\left(-i f_{\text{aux3}}\right) \tag{3.14}$$

$$f_{\text{trans}}(T, \phi, \Delta f, f_0, n_1, n_2, \alpha) = -\sqrt{T}\exp\left(i\left(\frac{\pi}{2} + \frac{1}{2}\left(f_{\text{aux2}} + f_{\text{aux3}}\right)\right)\right) \tag{3.15}$$

where the auxiliary functions $f_{\text{aux1}}$, $f_{\text{aux2}}$ and $f_{\text{aux3}}$ have been defined for better readability. For mirrors we would set $\alpha = 0$. We combine these three functions into one surface function which first calculates the reflection and transmission coefficients $R$ and $T$ from the reflectivity fraction $R_f$ and the surface loss $L$ and then computes the respective entries of the coupling matrices.

The combination of laser, space and surface function is sufficient to calculate all entries of the carrier matrix. Fig. 3.2 shows the simulation step for the cavity setup from code block

3.1. First, Differometor updates the parameter array with the values of the optimized parameters. Secondly, it expands this parameter array into a second dimension by the values of the changing parameters. From here on, this second dimension of the parameter array is processed in parallel using the JAX feature vmap. Third, Differometor selects the necessary functions for the optical setup at hand and applies them to the corresponding function inputs by making use of the JAX features vmap and lax.switch (steps 3, 4 and 5 in Fig. 3.2). The resulting output array is filtered from dummy values and the returned values get placed in the interferometer matrix which holds both carrier matrix and input vector (steps 6 and 7 in Fig. 3.2). After separating these two arrays, Differometor solves the system of equations by making use of jax.numpy.linalg.solve (steps 8 and 9 in Fig. 3.2).

### 3.3.2 The Signal System

The signal system is responsible for the propagation of signal sidebands that potentially get created by e.g. gravitational waves modulating the arm length of an interferometer. We can extend the setup from code block 3.1 to include a gravitational wave signal which is modulating the cavity:

```
import networkx as nx

G = nx.DiGraph()
G.add_node("l0", component="laser")
G.add_node("m1", component="mirror", properties={"reflectivity": 0.99})
G.add_node("m2", component="mirror", properties={"reflectivity": 0.991})

G.add_edge("l0", "m1")
G.add_edge("m1", "m2", properties={"length": 1})

G.add_node("f", component="frequency")
G.add_node("s0", component="signal", target="m1_m2")

G.add_node("reflected_power", component="detector", target="m1",
↪  direction="out")
G.add_node("circulating_power", component="detector", target="m2")
G.add_node("transmitted_power", component="detector", target="m2",
↪  port="right", direction="out")
```

Code 3.2: The definition of a cavity setup with gravitational wave signal in Differometor.

Here we add a frequency component similar to the $fsig$ command in FINESSE and apply a signal with default properties to the space between the two cavity mirrors.

We described the core equations behind the signal system in section 2.3.3. Differometor implements them by defining two more functions. First, the signal function

$$f_{\text{signal}}(h, \varphi) = h \exp(\mathrm{i}\varphi), \tag{3.16}$$

implementing the amplitude and phase of a signal, e.g. a gravitational wave as described in equation 2.12. The space modulation function then implements the remaining signal parts of the equations 2.54 and 2.55:

$$f_{\text{mod}}(f_s, L, n) = -\mathrm{i}\frac{f_0}{2f_s} \sin\left(\frac{2\pi f_s L n}{2c}\right) \exp(-\mathrm{i}\frac{2\pi f_s L n}{2c}). \tag{3.17}$$

Here, $f_0$ is the default laser frequency (a user defined constant in Differometor), $f_s$ is the signal frequency, $L$ is the macroscopic distance of the space and $n$ is the refractive index of the space. In contrast to the interferometer matrix of a pure carrier system (e.g. given for a cavity setup in equation 2.28), the signal matrix has one additional row and column:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -r_1 & 1 & -it_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & f_\text{s} & 0 & 0 & f_\text{mod} \cdot E_{c3} \\ -it_1 & 0 & -r_1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & f_\text{s} & 1 & 0 & 0 & 0 & f_\text{mod} \cdot E_{c5} \\ 0 & 0 & 0 & 0 & -r_2 & 1 & -it_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -it_2 & 0 & -r_2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \\ E_6 \\ E_7 \\ E_8 \\ S_1 \end{pmatrix} = \begin{pmatrix} E_{1i} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ E_{7i} \\ 0 \\ f_\text{sig} \end{pmatrix} \tag{3.18}$$

Here, $f_\text{s}$ is the output of the space function from section 3.3.1 and $f_\text{mod}$ and $f_\text{sig}$ are the signal and modulation function outputs. $E_{c3}$ and $E_{c5}$ are solutions from the carrier system. Analogous to FINESSE, the signal gets therefore injected into the space between the cavity mirrors and gets multiplied with the underlying carrier field to cover all parts of the equations 2.54 and 2.55.

As the signal system introduces new parameters (e.g. signal frequency, amplitude and phase), it also comes with additional index and value arrays of signal system parameters which change during simulation, just as in step 2 of the carrier system in Fig. 3.2.

The only other difference to the carrier system is the simulation of both upper and conjugated lower sidebands. This doubles the size of the interferometer matrix as all the matrix entries have to be calculated for two different frequencies. Fig. 3.3 shows the structure of the resulting signal interferometer matrix. Besides that, the simulation step of the signal system works exactly like the one of the carrier system described in section 3.3.1.

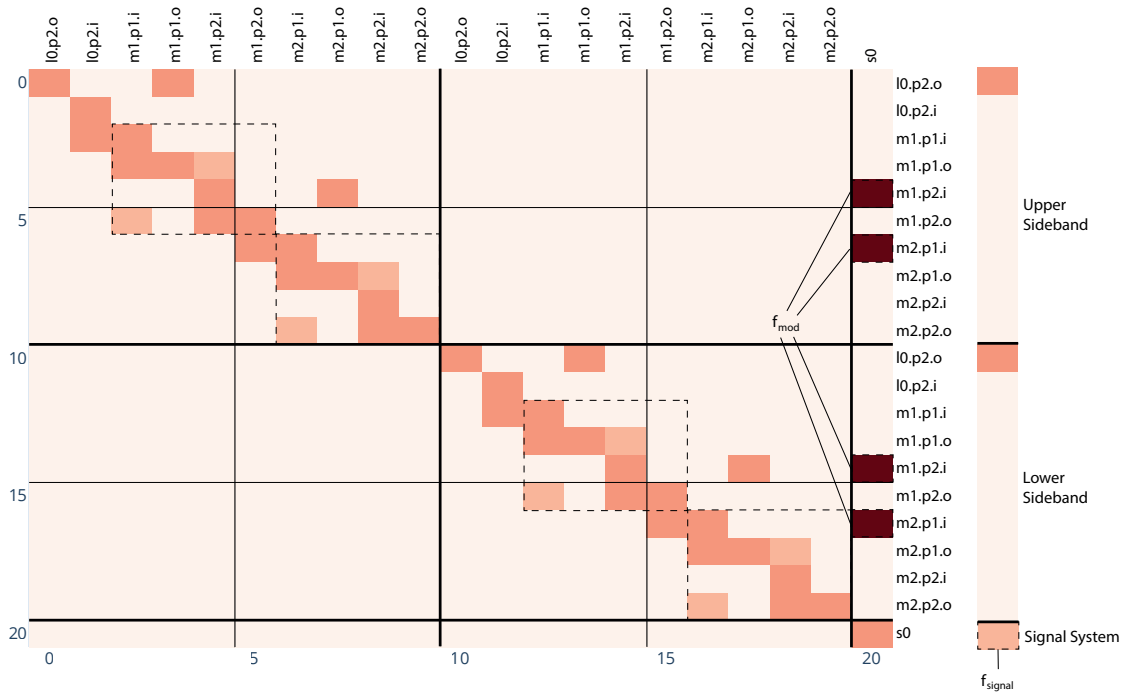


Figure 3.3: The structure of the signal system interferometer matrix for the cavity setup from code block 3.1. p1 and p2 refer to left and right ports respectively. i and o refer to input and output port. Colors indicate the magnitude of the complex entries. The matrix is divided into upper sideband, lower sideband and signal system. The mirror coupling matrices in upper and lower sideband are marked by the dashed boxes. The signal system adds one new row and column for each signal. The entries from equations 3.16 and 3.17 are labeled by their name.

### 3.3.3 Quantum Noise

The simulation step for quantum noise implements equation 2.75 which includes the matrix multiplications from equation 2.74. So besides the signal interferometer matrix $\mathbb{M}$ from section 3.3.2, the build step now also prepares information about the input noise matrix $\mathbb{V}_i$. Similar to carrier and signal system, the simulation step receives the input noise matrix of the right shape initialized with zeros and a set of arrays with instructions of how to update this matrix. Again, the simulation step uses similar arrays and a similar

update mechanism as depicted in Fig. 3.2. The list of available functions gets extended by three new input noise functions. First, the vacuum noise function to calculate the input noise for lasers and unused component ports:

$$f_{\text{vacuum\_noise}}() = \frac{u}{2}, \tag{3.19}$$

Similar to the global unit vacuum constant in FINESSE, $u$ is a custom vacuum unit that can be modified by the user of Differometor and is set to 1 by default. The second input noise function is the loss noise function for surface components with loss:

$$f_{\text{loss\_noise}}(L) = \frac{L}{2}. \tag{3.20}$$

Here, $L$ is the loss of the respective surface for which the input noise gets calculated. Lastly, another input noise function is the squeezer function for injecting squeezed vacuum into an optical system, implementing equation 2.76:

$$f_{\text{diagonal}}(r, \phi) = u \cosh(2r) \tag{3.21}$$

$$f_{\text{off\_diagonal}}(r, \phi) = u \sinh(2r) \exp(2i\phi) \tag{3.22}$$

$$f_{\text{squeezer}}(r, \phi) = [f_{\text{diagonal}}, f_{\text{off\_diagonal}}, f^*_{\text{diagonal}}, f^*_{\text{off\_diagonal}}] \tag{3.23}$$

where $r$ and $\phi$ are the squeezing parameter and the squeezing angle respectively. The build step then again passes function input indices, function indices, function output indices and matrix indices (see step 3, 4, 6 and 7 from Fig. 3.2) to the simulation step, which in turn uses these to update the input noise matrix.

We can extend the setup in code block 3.2 with an additional quantum noise detector and a squeezer which injects squeezed vacuum into the input port on the left side of mirror 2:

```python
import networkx as nx

G = nx.DiGraph()
G.add_node("l0", component="laser")
G.add_node("m1", component="mirror", properties={"reflectivity": 0.5, "loss": 0.5})
G.add_node("m2", component="mirror", properties={"reflectivity": 0.5, "loss": 0.5})

G.add_edge("l0", "m1")
G.add_edge("m1", "m2", properties={"length": 1})

G.add_node("f", component="frequency")
G.add_node("s0", component="signal", target="m1_m2")

G.add_node("sq1", component="squeezer", target="m2", properties={"db": 10, "angle": 90})

G.add_node("reflected_power", component="detector", target="m1", direction="out")
G.add_node("circulating_power", component="detector", target="m2")
G.add_node("transmitted_power", component="detector", target="m2", port="right", direction="out")

G.add_node("noise", component="qnoised", target="m2", port="right", direction="out")
```

Code 3.3: The definition of a cavity setup with an additional quantum noise detector, a squeezer and a gravitational wave signal in Differometor.

We also changed the reflectivities of the two mirrors and added some loss to demonstrate the resulting input noise entries. Fig. 3.4 shows the resulting input noise matrix. The noise entries from the squeezer in rows 6 and 16 and columns 6 and 16 connect upper and lower sidebands and indicate the correlation between the resulting sidebands. Apart from that, the other ports are filled with normal vacuum noise or quantum noise originating from the loss of the mirrors.

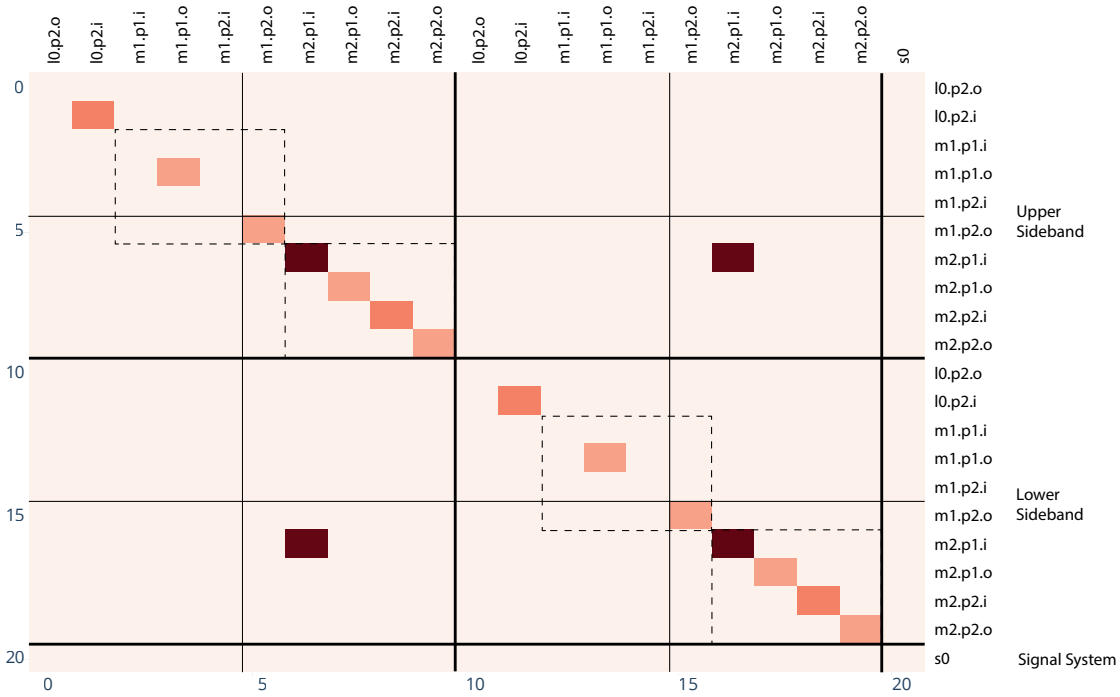


Figure 3.4: The structure of the input noise matrix for the setup defined in code block 3.3.

As described with equation 2.75, the quantum noise gets calculated individually for each photodetector using a selection vector $\vec{s}_c$ which is the solution of the carrier system reduced to only one non-zero entry at the position of the detector. With all these matrices and vectors, the quantum noise simulation step of Differometor then performs the matrix multiplications necessary to calculate equation 2.75, solving the equations for all photodetectors in parallel by making use of the JAX feature vmap. In the end, Differometor scales the noise output with the Planck constant, and the default frequency.

### 3.3.4 Optomechanics

We can extend the setup from code block 3.2 and add free masses with default mass 40 kg to the two mirrors to allow for optomechanical effects:

```python
import networkx as nx

G = nx.DiGraph()
G.add_node("l0", component="laser")
G.add_node("m1", component="mirror", properties={"reflectivity": 0.99})
G.add_node("m2", component="mirror", properties={"reflectivity": 0.991})

G.add_node("m1_mass", component="free_mass", target="m1")
G.add_node("m2_mass", component="free_mass", target="m2")

G.add_edge("l0", "m1")
G.add_edge("m1", "m2", properties={"length": 1})

G.add_node("f", component="frequency")
G.add_node("s0", component="signal", target="m1_m2")

G.add_node("reflected_power", component="detector", target="m1",
↪  direction="out")
G.add_node("circulating_power", component="detector", target="m2")
G.add_node("transmitted_power", component="detector", target="m2",
↪  port="right", direction="out")
```

Code 3.4: The definition of a cavity setup with gravitational wave signal and optomechanical effects in Differometor.

The Differometor simulation step for optomechanics is similar to the signal system simulation step in that it only adds new rows and columns to the interferometer matrix and extends the list of available functions. Specifically, the first new function is the implementation of equation 2.81, taking the signal frequency $f_s$ and the mass $m$ of the corresponding surface as inputs and returning the mechanical susceptibility for a free-floating surface:

$$f_{\text{suscept}}(f_s, m) = -\frac{1}{m(2\pi f_s)^2}. \tag{3.24}$$

The optomechanics implementation also introduces new functions to calculate the radiation pressure forces from both sides of a surface. Both functions are implementations of equation 2.77 and their sign implements the opposing momentum from equation 2.78:

$$f_{\text{force1}}(\alpha) = -\frac{\cos\alpha}{cx} \tag{3.25}$$

$$f_{\text{force2}}(\alpha) = \frac{f_{\text{aux1}}}{cx}. \tag{3.26}$$

Here, $\alpha$ is again the potential angle of incidence of the surface at hand and $f_{\text{aux1}}$ is the auxiliary function from equation 3.10. $x$ is another user defined Differometor constant (again equivalent to FINESSE) to scale mechanical output signals and is set to $10^{-9}$ by default. $P(\omega)$ from equation 2.77 enters the equation later through the structure of the equation system defined by the interferometer matrix. In addition, after placing the return values of $f_{\text{force1}}$ and $f_{\text{force2}}$ in the interferometer matrix, they get multiplied with the corresponding carrier field from the respective surface port.

To close the loop from radiation pressure through existing fields to the creation of new sidebands as described in section 3.3.4, only the functions implementing the resulting phase shift from equation 2.82 are still missing:

$$f_{\text{correct}}(f_s, \phi) = \exp(i\phi\frac{f_s}{f_0}) \tag{3.27}$$

$$f_{\text{phase1}}(f_s, \phi, R, n_1, \alpha) = -ixf_{\text{correct}}\frac{2\pi}{\lambda}\cos\alpha f_{refl1} \tag{3.28}$$

$$f_{\text{phase2}}(f_s, \phi, R, n_1, n_2, \alpha) = ixf^*_{\text{correct}}\frac{2\pi}{\lambda}f_{\text{aux1}}f_{refl2} \tag{3.29}$$

$f_{\text{phase1}}$ and $f_{\text{phase2}}$ are the actual functions, $f_{\text{correct}}$ is defined for better readability. The functions take the signal frequency $f_s$, the surface tuning $\phi$, the reflectivity $R$, the refractive indices $n_1$ and $n_2$ and the angle of incidence $\alpha$ as input. The reflectivity $R$ is calculated beforehand from the reflectivity fraction $R_f$ and the surface loss $L$. $f_{\text{refl1}}$, $f_{\text{aux1}}$ and $f_{\text{refl2}}$ are the functions from equation 3.13, 3.10 and 3.14 respectively. $\lambda$ is the default wavelength in Differometor which is set to $1064 \cdot 10^{-9}$ m. $x$ is again the scaling factor for mechanical output signals.

Compared to equation 2.82, $f_{\text{phase1}}$ and $f_{\text{phase2}}$ don't contain $\delta z(\omega)$. This surface displacement enters the equation again later through the structure of the equation system defined by the interferometer matrix. Fig. 3.5 shows this matrix structure and indicates where all the return values from these functions will be placed. Mirror $m1$ occupies rows 2 to 5 and columns 2 to 5 for the upper sideband and rows 12 to 15 and columns 12 to 15 for the lower sideband. For each mirror port, the corresponding radiation pressure forces get calculated via the results from $f_{\text{force1}}$ and $f_{\text{force2}}$ which have been placed in row 20 and multiplied by the carrier fields from the respective mirror ports. By summing over

all these entries multiplied with the solutions for the signal fields at the respective mirror ports, equation 2.78 gets implemented. Here, upper and lower sidebands get added together to calculate the field power according to equation 2.79, leaving e.g. quantum noise sidebands in a correlated state.

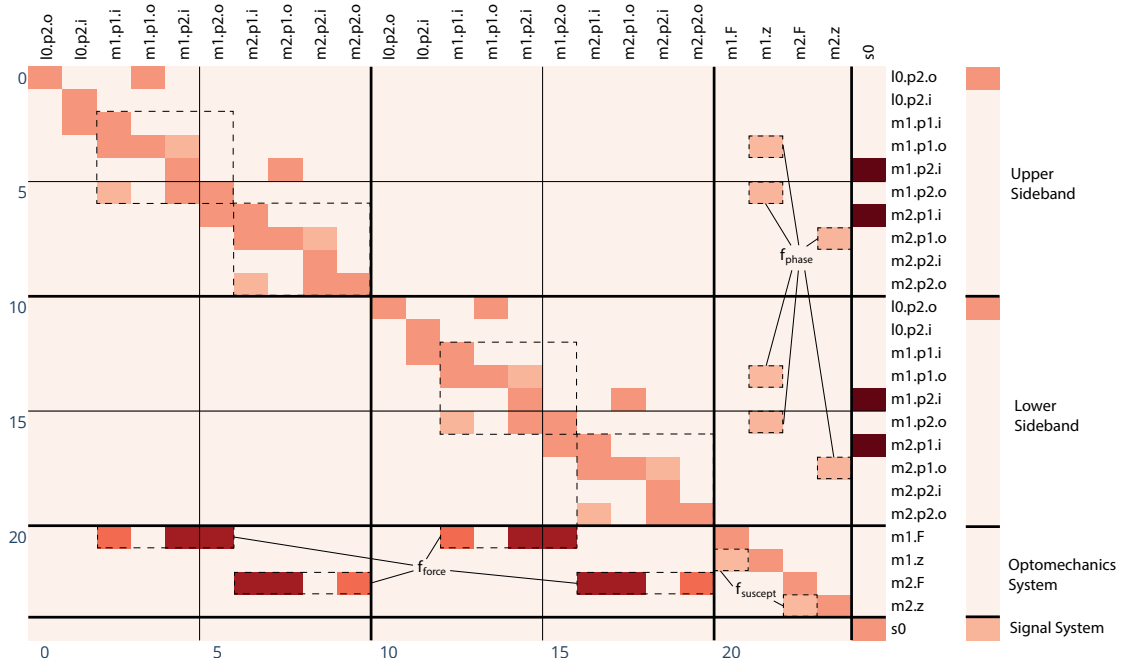


Figure 3.5: The structure of the signal system interferometer matrix for the cavity setup from code block 3.4. p1 and p2 refer to left and right ports respectively. i and o refer to input and output ports. F and z refer to force and susceptibility entries respectively. Colors indicate the magnitude of the complex entries. The matrix is divided into upper sideband, lower sideband, optomechanics system and signal system. The mirror coupling matrices in upper and lower sideband are marked by the dashed boxes. The optomechanics system adds 2 new rows and columns for each free mass mirror. The entries of the optomechanics system are labeled by the computing functions from equations 3.29, 3.26 and 3.24.

Through the susceptibility factor in row 21, column 20, this total force then couples into the calculation of the new sidebands carried out by the entries in column 22. These are formed by placing the return values from $f_{\text{phase1}}$ and $f_{\text{phase2}}$ and multiplying them with the carrier fields from the respective input ports. $f_{\text{phase1}}$ and $f_{\text{phase2}}$ include the potential angle of incidence as well as the computation of the reflection and tuning phase of the carrier field by $f_{\text{refl1}}$ and $f_{\text{refl2}}$. As the frequency shifted sidebands will accumulate a slightly different detuning phase when coming back from the surface, $f_{\text{phase1}}$ and $f_{\text{phase2}}$ also contain $f_{\text{correct}}$ to account for this phase difference.

This optomechanics system enables the simulation of simple radiation pressure effects that can be observed in the resulting sensitivity curve.

## 3.4 Differometor vs. Finesse

In this section, we compare Differometor to FINESSE in terms of functionality, accuracy and simulation time performance. The functionality comparison in section 3.4.1 provides a better understanding of what Differometor can and cannot do at the moment. The accuracy comparison in section 3.4.2 builds the basis for future trust in this new simulation tool and the simulation time performance comparison in section 3.4.3 demonstrates that Differometor already offers some advantages when running pure simulations without any optimizations.

### 3.4.1 Functionality

Just like FINESSE, Differometor is a frequency domain simulator that only works under the assumption of a steady state optical setup. It can therefore not be used to simulate nonstationary and nonlinear processes such as lock acquisition, control systems or time evolutions of light fields inside cavities. For the purpose of digital discovery of interferometer setups in a steady state, we can use Differometor to find possible setups and then outsource the process of locking these setups into a steady state to other simulation tools like E2E [71].

So far, Differometor implements only a subset of the functionality offered by FINESSE. We focused on only those features necessary to reproduce the digital discovery experiments conducted by Krenn et al. [6]. Thus, all Differometor simulations are based on a plane-wave approximation. This allows for a wide range of different simulations, but does not support more complex analysis tasks involving the beam shape and position, e.g. the effects of misaligned components. In addition to a plane-wave approximation, FINESSE also offers the functionality to extend simulations using transverse electromagnetic modes (TEM) describing the transformation of the spatial distribution of a light beam perpendicular to the optical axis [8]. Specifically, FINESSE implements Hermite-Gaussian beams of different order. This allows for the simulation of e.g. thermal distortions, tilted surfaces, aperture effects or component misalignment. In addition, FINESSE offers functionality to simulate the surface motions of a particular surface using Finite-Element-Model (FEM) software. In combination with Hermite-Gaussian beams, this allows for simulation of parametric instabilities where the surface motion positively or negatively feedbacks itself due to optical coupling.

When it comes to the simulation of optomechanical effects, Differometor only implements longitudinal surface motion along the optical axis. Here, FINESSE also offers rotational

yaw and pitch motions allowing it to simulate e.g. torsional optical springs. While Differometor only offers the free mass as a suspension simulation, FINESSE also offers more complex suspension systems like pendulums or customised suspension elements.

FINESSE also offers functionality to simulate simple thermal effects. When a beam transmits through or reflects from an optical component, a small fraction of power gets absorbed by the element. This leads to a thermal gradient within the component which can cause changes of the refractive index (thermo-refractive effect) and can expand and deform the optics (thermo-elastic effect). FINESSE implements this using the Hello-Vinet equations, a collection of analytic solutions for on-axis cylindrically symmetric beams interacting with a cylindrical mirror. This allows for the simulation of e.g. thermal lensing and deformations.

For the digital discovery tasks of new interferometer layouts that Differometor is designed for right now, all these additional features are not yet crucial and will only be added in future Differometor versions if necessary.

```python
import networkx as nx

def convert(transmissivity, loss, phi):
    reflectivity = 1 - transmissivity / (1 - loss)
    return {"reflectivity": reflectivity, "loss": loss, "tuning": phi}

G = nx.DiGraph()
G.add_node("L0", c="laser", p={"power": 125})
G.add_node("bs", c="beamsplitter", p={"reflectivity": 0.5, "alpha": 45})
G.add_node("prm", c="mirror", p=convert(0.03, 37.5e-6, 90))
G.add_node("itmx", c="mirror", p=convert(0.014, 37.5e-6, 90))
G.add_node("etmx", c="mirror", p=convert(5e-6, 37.5e-6, 89.999875))
G.add_node("itmy", c="mirror", p=convert(0.014, 37.5e-6, 0))
G.add_node("etmy", c="mirror", p=convert(5e-6, 37.5e-6, 0.000125))
G.add_node("srm", c="mirror", p=convert(0.2, 37.5e-6, -90))
G.add_node("sq1", c="squeezer", p={"db": 10, "angle": 90})
G.add_node("itmxsus", c="free_mass", target="itmx")
G.add_node("etmxsus", c="free_mass", target="etmx")
G.add_node("itmysus", c="free_mass", target="itmy")
G.add_node("etmysus", c="free_mass", target="etmy")

G.add_edge("L0", "prm")
G.add_edge("prm", "bs", p={"length": 53})
G.add_edge("bs", "itmx", p={"length": 4.5})
G.add_edge("itmx", "etmx", p={"length": 3995})
G.add_edge("bs", "itmy", source_port="top", p={"length": 4.45})
G.add_edge("itmy", "etmy", p={"length": 3995})
G.add_edge("bs", "srm", source_port="bottom", p={"length": 50.525})
G.add_edge("sq1", "srm", target_port="right")

G.add_node("f", c="frequency", p={"frequency": 5})
G.add_node("darmx", c="signal", target="itmx_etmx")
G.add_node("darmy", c="signal", target="itmy_etmy", p={"phase": 180})

G.add_node("qnoised", c="qnoised", target="srm", port="right", d="out")
G.add_node("pd1", c="detector", target="srm", port="right", d="out")
```

Code 3.5: Defining the simplified aLIGO setup from Fig. 2.3 in Differometor. c=, p= and d= stand for component=, properties= and direction= respectively and are only introduced for better readability.

### 3.4.2 Accuracy Analysis

To validate Differometor, we conduct benchmark simulations against the established FINESSE simulator from section 2.3.7, which has itself been validated against other interferometer simulation software, analytical solutions and against experimental data [45, 90, 86]. Fig. 3.6 shows comparisons between the Differometor and FINESSE sensitivity curves calculated for the simplified aLIGO setup from Fig. 2.3 and code block 3.5 when changing different parameters. The aLIGO setup includes all possible mechanisms and features of Differometor, from the carrier system and the simulation of a gravitational wave signal to optomechanical effects and quantum noise sidebands.

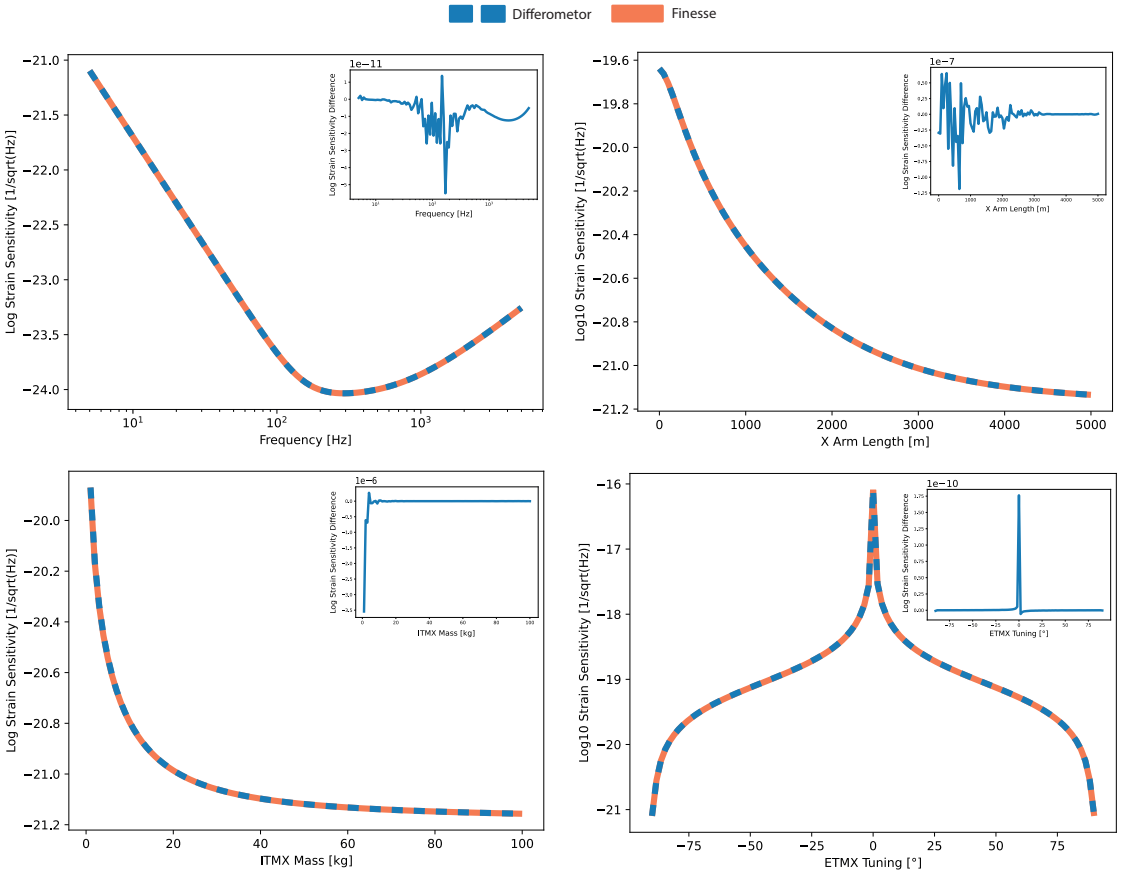


Figure 3.6: Comparison of sensitivity curves computed by Differometor and FINESSE for the simplified aLIGO setup from Fig. 2.3 and code block 3.5 for different changing parameters. The inlets show the difference between the two curves. For each plot, 100 values of the respective changing parameter have been simulated.

For quantitative assessment of the agreement, we compute the normalized root-mean-

square deviation (NRMSD) for each curve over the 100 values of the respective changing parameter using

$$\text{NRMSD} = \frac{\sqrt{\sum_{i=1}^{N} \left(log_{10}\left(y_{i,\text{Differometor}}\right) - log_{10}\left(y_{i,\text{Finesse}}\right)\right)^2 / N}}{\sum_{i=1}^{N} \left(-log_{10}\left(y_{i,\text{Differometor}}\right)\right) / N} \times 100\% \tag{3.30}$$

where $y$ represents the curve being compared and $N$ is the number of values for the changing parameter. All comparisons show excellent $\ll 1\%$ agreement with maximum deviations on the order of $10^{-7}$. These can be explained by the numerical accuracy of JAX vs. NumPy. JAX by default enforces single-precision numbers as XLA, the layer between frontend machine learning frameworks and hardware backends, does not support 64-bit precision on all backends like e.g. GPU and TPU.

This demonstrates the close agreement of Differometor with the established interferometer simulator Finesse for complex simulation tasks and also validates the accuracy of the individual sub-components like the carrier, signal, optomechanics and noise systems.

### 3.4.3 Performance Analysis

The auto-differentiation feature of JAX is expected to enable a large speedup in gradient calculation when running optimizations. However, JIT compilation and GPU execution should also improve the simulation time performance even without running optimizations with gradient calculations.

To confirm this, we run simulations of the simplified aLIGO setup from Fig. 2.3 and code block 3.5 with the signal frequency as changing parameter. We then vary the size of the value array for the signal frequency from 1 to $2 \cdot 10^5$. This corresponds to the $V$ dimension in Fig. 3.2 and means that in each run, the setup is simulated for $V$ different frequency values in parallel. We thus increase the number of simulations done in each run and measure the time it takes to perform the build step and the simulation step respectively. For Finesse, we define the build step as everything that comes before the finesse.Model.run command which is mainly the parsing of the setup definition into a Finesse model. The simulation step is then everything that happens within the finesse.Model.run command, which is the actual simulation of the setup. For Differometor, the build and simulation steps are explained in sections 3.2 and 3.3. We average the timings for each step over 10 iterations where each run performs one iteration before the time measurement starts to exclude any JIT compilation and caching effects. CPU runs are done on Intel Xeon Gold 6130 x86_64 Skylake-SP CPUs while GPU runs are done on Nvidia Quadro RTX 6000 Turing GPUs. Fig. 3.7 shows the results of this experiment.

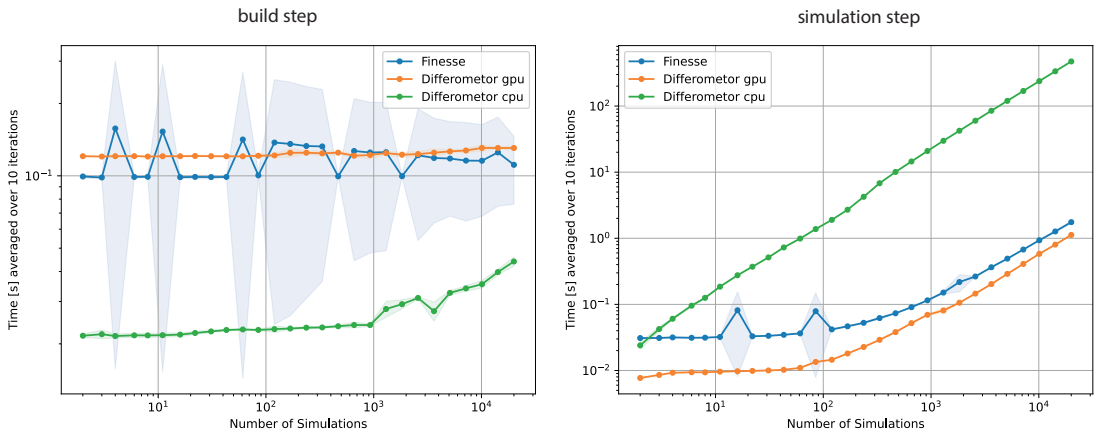


Figure 3.7: Comparison of average computation time for build step and simulation step for the simplified aLIGO setup from Fig. 2.3 and code block 3.5. The number of simulations corresponds to the size of the value array of the frequency as the changing parameter ($V$ in Fig. 3.2). The build step of FINESSE is the parsing of the defined setup. Shaded areas indicate the standard deviation.

For the build step, that is processing the defined setup and compiling the set of index and value arrays necessary for the simulation step, we observe that the Differometor GPU runs perform similar to FINESSE while the Differometor CPU runs consistently outperform the FINESSE build step. As mentioned in section 3.2, the build step is not implemented in JAX, but in NumPy and only converts all arrays to JAX at the end. The discrepancy between the GPU and CPU runs for Differometor is due to the fact that the GPU runs have to shift all arrays from CPU to GPU at the end of the build step. The FINESSE build step includes some computational overhead due to the parsing process of the self defined syntax. In general, the build step only needs to be performed once for each setup, followed by a larger number of simulations performed with the compiled model or arrays. Thus, the more interesting timing comparison is the one of the simulation step shown in Fig. 3.7 on the right.

Here, we first observe that the Differometor GPU runs outperform the Differometor CPU runs. This is expected due to the parallelization advantages of running matrix operations on a GPU instead of a CPU. Differometor simulations on GPU also consistently outperform FINESSE simulations. This confirms the expected speedup due to JIT compilation, GPU execution and automatic parallelization with JAX *vmap*. Being implemented in NumPy, FINESSE cannot make use of these features, resulting in slightly slower simulations.

When profiling the different steps of the simulation process shown in Fig. 3.2, we observe that the steps 3 to 7 take the maximum amount of time and present the current

computational bottleneck of Differometor. JAX performs array updates like the one in Fig. 3.2 step 7 mainly out-of-place by returning a new version of the modified array instead of altering the existing one. Reducing the size of the modified interferometer matrix using sparsification promises another performance boost for future Differometor versions.

Fig. 3.7 also shows that Differometor simulations on CPU are performing worse than the FINESSE counterpart which implements sparsification and supports in-place updates. After implementing the sparsification of interferometer matrices in Differometor, the CPU version is expected to perform comparably to FINESSE.

Fig. 3.7 confirms that Differometor simulations with GPU already offer some performance advantages, even without running optimizations and gradient calculations.

# Chapter 4

# Differometor for Digital Discovery

In this chapter we apply our new Differometor simulator to first digital discovery tasks and evaluate its performance with respect to gradient evaluation and convergence. In section 4.1, we present an optimization of the simplified aLIGO setup from Fig. 2.3 and code block 3.5, where Differometor starts from a setup with randomized parameters and then rediscovers a version of the setup that produces the same sensitivity curve. We compare the optimization performed by Differometor and AdamW [131] to a similar optimization with FINESSE and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [114, 115, 116, 117] to demonstrate the effect of auto-differentiation on gradient evaluations.

In section 4.2 we extend this experiment by putting constraints on the power applied to optical components and detectors and search the parameter space for a setup that outperforms the simplified aLIGO setup with respect to the sensitivity curve.

Finally, in section 4.3, we implement the quasi-universal interferometer (UIFO) setup described in section 2.4.2 and demonstrate the scalability of Differometor by running the same optimizations as in section 4.2 with UIFO setups of different sizes.

All optimizations in this chapter are performed on Intel Xeon Gold 6130 x86_64 Skylake-SP CPUs or Nvidia Quadro RTX 6000 Turing GPUs.

## 4.1 Rediscovery of Advanced LIGO

In section 3.4.3, we have shown that Differometor simulations offer performance advantages due to GPU execution and JIT compilation. Another JAX feature that is expected to improve the performance of Differometor optimizations is auto-differentiation. To demonstrate this performance advantage, we start with the simplified aLIGO setup from Fig. 2.3 and code block 3.5. This setup has 21 components with 46 parameters. As surface losses should always be chosen as small as possible, we exclude these from the optimizations. To keep the optimization similar to the ones performed by Krenn et al.

[6], we also exclude refractive indices from the optimizations and are left with 31 optimized parameters. Table A.1 in the appendix lists all parameters included in this simplified aLIGO setup and indicates all optimized parameters.

We aim to reproduce the sensitivity curve with radiation pressure from Fig. 2.3 over 100 logarithmically distributed frequency values in the frequency range from 20 Hz to 5000 Hz, starting from a setup similar to the one in code block 3.5, but with all optimized parameters randomly initialized. The loss function to achieve this is the mean squared error:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} \left( \log_{10} y_i - \log_{10} \hat{y}_i \right)^2 , \tag{4.1}$$

where $N$ is 100, $y_i$ are the sensitivity values from the optimized setup and $\hat{y}_i$ are the sensitivity values from the setup in code block 3.5. The $\log_{10}$ is applied to account for the fact that the sensitivity curve spans multiple orders of magnitude (see Fig. 2.3), where the contribution of sensitivity values at small frequencies would be too large compared to sensitivity values at higher frequency ranges.

We select the optimization procedure based on empirical tests and inspired by the methods used by Krenn et al [6]. We initialize all parameters uniformly distributed between $-6$ and 6. Before each simulation, we then map all parameter values into the range 0 to 1 using a sigmoid function and scale them to their respective ranges shown in table 4.1. Here, we keep tuning parameters only between 0° and 90° as larger ranges lead to a strong decrease in the rate of successful optimizations. Other parameter ranges are chosen similar to what was used by Krenn et al. [6].

| **Parameter** | **Bounds (Min)** | **Bounds (Max)** |
|---|---|---|
| db [db] | 0.01 | 20 |
| angle [°] | -180 | 180 |
| power [W] | 0.01 | 200 |
| tuning [°] | 0 | 90 |
| mass [kg] | 0.01 | 200 |
| length [m] | 1 | 4000 |
| reflectivity [%] | 0 | 1 |
| phase [°] | -180 | 180 |
| alpha [°] | -180 | 180 |

Table 4.1: Physical parameter ranges for the optimization of the simplified aLIGO setup from Fig. 2.3 and code block 3.5.

To evaluate the optimization performance at different numbers of optimized parameters, we first start with only one optimized parameter and then consecutively add more until we reach the full set of 31 optimized parameters. We choose the optimized parameters in the order they appear in table A.1 (i.e. starting with the squeezing magnitude of squeezer sq1). For each set of optimized parameters, we first generate 1000 setups where the optimized parameters are randomly initialized while all other parameters are initialized as described in code block 3.5. We then calculate the loss of each of these setups and take the best performing setup as the starting point for our optimizations. This should enable the optimization to already start in a preferable region of the loss landscape where it can then improve by the following gradient updates.

In this comparison, we use the AdamW [131] optimizer from Optax [132] with learning rates 0.1 and 0.5 together with gradient clipping with maximum global norm 1.0 [133]. We choose two different learning rates to demonstrate the effect of the learning rate on the convergence time and success rate. It should be noted that the choice of optimizer is not the focus of this experiment and stems purely from a previous, empirical and non-exhaustive hyperparameter testing session. Weight decay might not be beneficial in this case and a pure Adam optimizer might be a better choice than AdamW.

In addition to the optimizations with Differometor and AdamW, we also use the optimization settings from the recent work by Krenn et al. [6] on the digital discovery of gravitational wave detectors as a baseline and perform simulations with Finesse and the BFGS optimizer. The procedure is the same as before, the only difference is the optimizer which is now run with standard BFGS hyperparameters from SciPy [134]. For the Differometor case, we run the optimizations until they either exceed 15.000 iterations or until their loss reaches below the loss threshold of $10^{-5}$. For FINESSE, we run the optimizations until they exceed 15.000 iterations, cross the loss threshold of $10^{-5}$ or until the BFGS step size reaches 0 after which running the optimization for longer won't result in meaningful changes.

Fig. 4.1 shows some example loss curves for successful Differometor and FINESSE optimizations for the maximum of 31 optimized parameters. Different from the example loss curves for unsuccessful optimizations in Fig. 4.2, all loss curves reach the loss threshold of $10^{-5}$. For learning rate 0.1, the successful optimizations reach the loss threshold after a sharp drop in the loss curve while optimizations with learning rate 0.5 often show oscillating behavior, indicating that the learning rate is too high. Successful BFGS optimizations reach the loss threshold after multiple phases of sharp drops and constant plateaus.

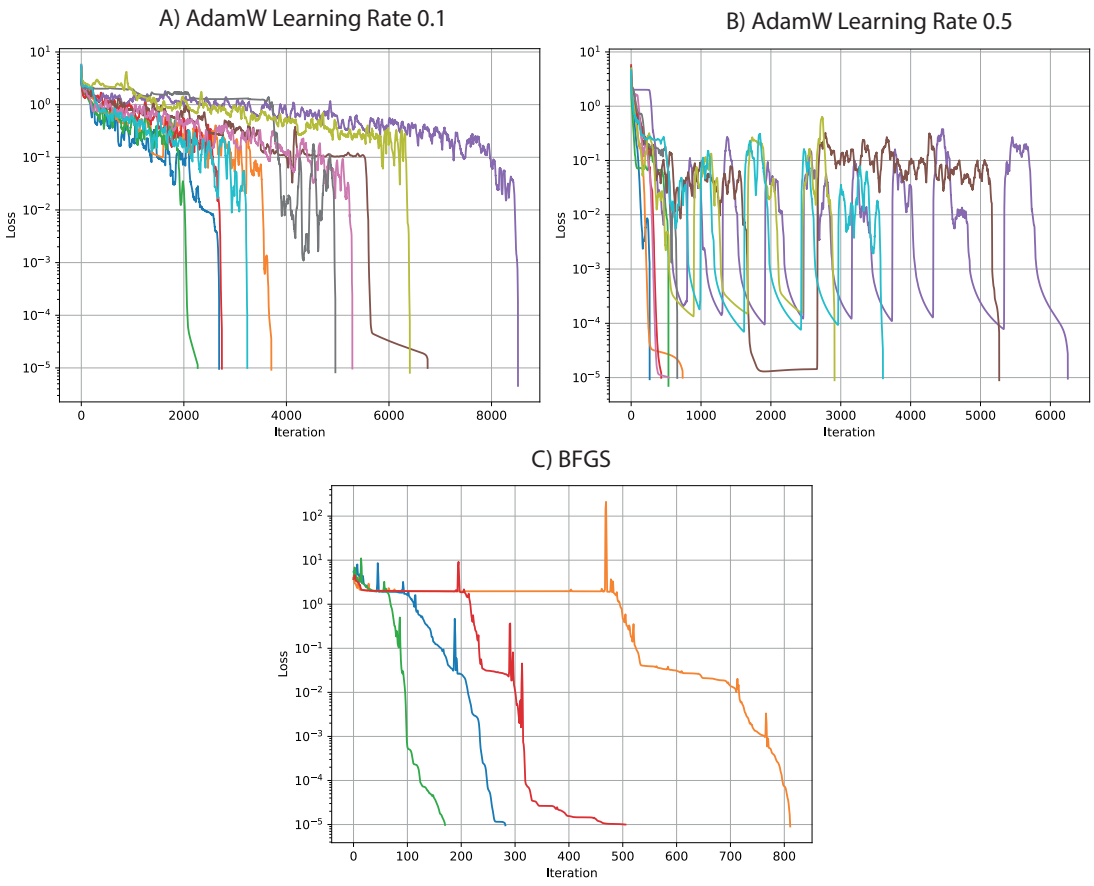


Figure 4.1: Examples of smoothed loss curves for successful Differometor and FINESSE optimizations for 31 optimized parameters. An optimization is successful if it crosses the loss threshold of $10^{-5}$ before being stopped by one of the termination criteria. An iteration is defined as a full gradient evaluation (in the case of BFGS this involves running multiple simulations).

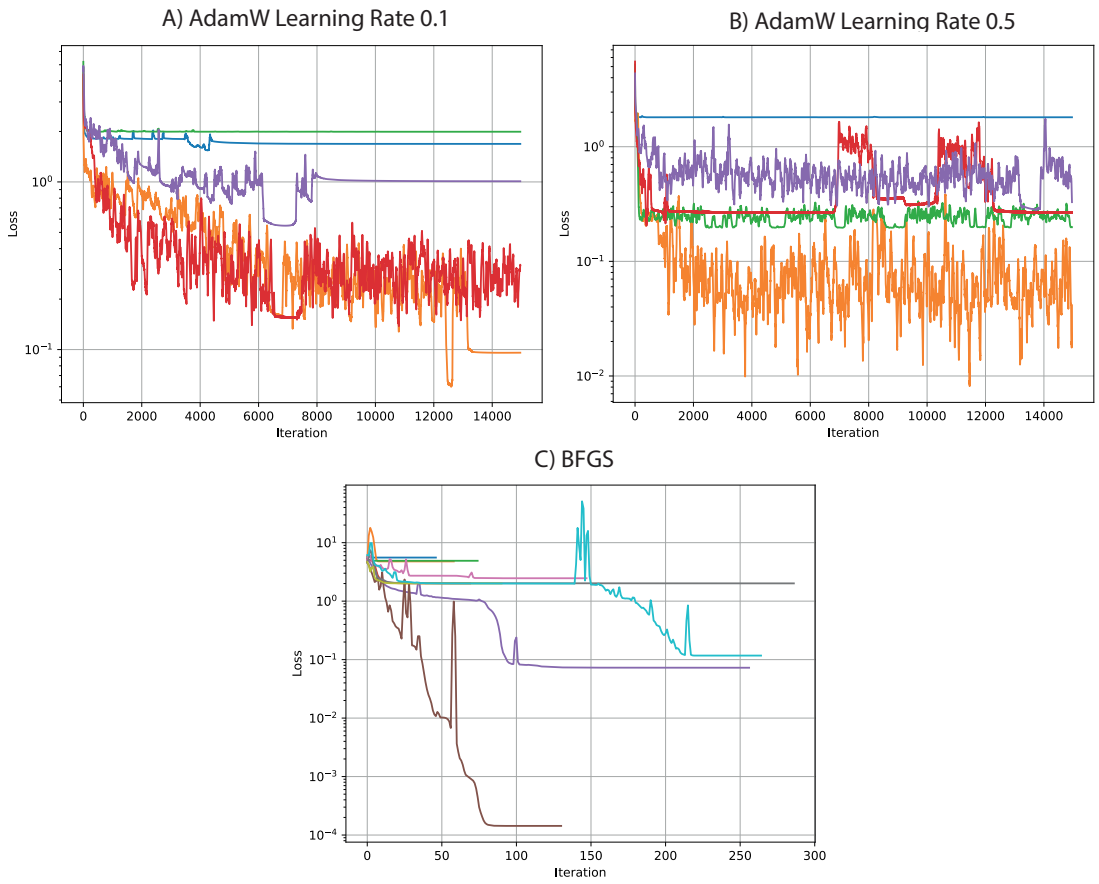


Figure 4.2: Examples of smoothed loss curves for unsuccessful Differometor and FINESSE optimizations for 31 optimized parameters. An optimization is unsuccessful if it doesn't cross the loss threshold of $10^{-5}$ before being stopped by one of the termination criteria. An iteration is defined as a full gradient evaluation (in the case of BFGS this involves running multiple simulations).

Fig. 4.3 A) shows the success rates (i.e. the number of successful runs) of all optimizations for the three different optimization combinations FINESSE with BFGS, Differometor with AdamW and learning rate 0.1 and Differometor with AdamW and learning rate 0.5. Here, a successful run is an optimization that crosses the loss threshold of $10^{-5}$ before being stopped by one of the termination criteria. For each number of optimized parameters there are 100 optimization runs where each run follows the optimization procedure described above.

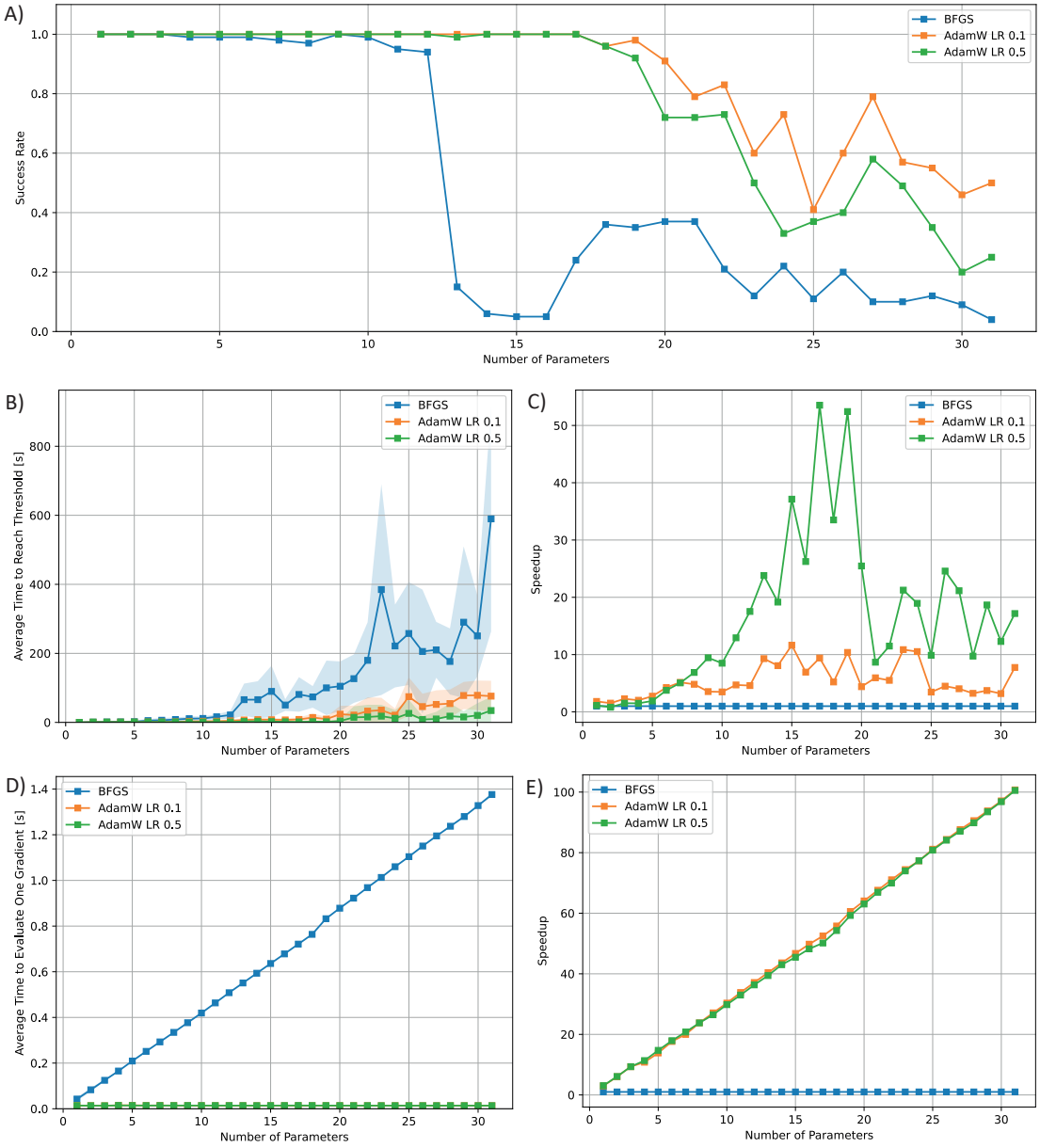


Figure 4.3: **A)** Success rates for the optimization combinations FINESSE with BFGS, Differometor with AdamW and learning rate 0.1 and Differometor with AdamW and learning rate 0.5. For each number of optimized parameters, the success rate states how many runs out of 100 cross the loss threshold of $10^{-5}$ before being stopped by one of the termination criteria. **B)** Convergence timings (the time until an optimization crosses the loss threshold of $10^{-5}$) averaged over all successful runs from A). **C)** Convergence time speedup as convergence time divided by the convergence time of BFGS optimizations. **D)** Gradient evaluation timings (convergence time divided by the number of iterations) averaged over all successful runs from A). **E)** Gradient evaluation time speedup as gradient evaluation time divided by the gradient evaluation time of BFGS optimizations.

For the two Differometor combinations, we observe that the success rate stays at almost 1 until reaching the 18th optimized parameter (i.e. the beam splitter tuning as the first tuning parameter, see table A.1) after which the success rates drop to around 0.5 for learning rate 0.1 and 0.25 for learning rate 0.5 after reaching 31 optimized parameters. The learning rate 0.1 consistently outperforms the learning rate 0.5 in terms of the success rate. For the FINESSE optimization, the success rate already drops significantly at the 13th optimized parameter (i.e. the first mirror mass for $itmx$) and reaches only 0.04 at 31 optimized parameters. This success rate behavior demonstrates that for different optimizers, parameters can have strongly different effects on the optimization outcome.

Optimizations with AdamW and auto-differentiation consistently outperform optimizations with BFGS and numerical gradient approximation in terms of success rate. This demonstrates the advantage of unlocking direct gradient calculation through implementing the simulator with auto-differentiation.

For the timings of convergence and gradient evaluation, we now restrict our comparison to include only successful runs that have crossed the loss threshold of $10^{-5}$. We measure the convergence time of the optimization until the loss threshold is crossed, excluding parsing and compilation times from the build and JIT steps. We then calculate the time to perform a single gradient evaluation as the convergence time divided by the number of iterations. We average all timings over all successful runs. Fig. 4.3 B) to E) show the results of these timing tests.

Optimization runs with learning rate 0.5 outperform runs with the learning rate of 0.1, indicating some speedup benefits despite the oscillating loss curves in Fig. 4.1 and the lower success rates. In terms of convergence time, Differometor optimizations with AdamW consistently outperform FINESSE optimizations using BFGS. Specifically for the gradient evaluation time we observe that the evaluation time to numerically approximate the gradient in BFGS increases linearly with the number of optimized parameters while the evaluation times for AdamW using auto-differentiation stay nearly constant. This leads to gradient evaluation speedup factors of up to 100 for 31 optimized parameters (see Fig. 4.3 E)). This demonstrates the expected advantage of using a differentiable simulator like Differometor compared to using FINESSE and being restricted to numerical gradient approximation.

However, comparing the speedups of convergence and gradient evaluation time, we observe that Differometor is not able to directly translate gradient evaluation speedups into convergence time speedups. When using the AdamW optimizer, the convergence time speedup factors vary strongly between 0 and 10 for learning rate 0.1 and 0 and 53 for learning rate 0.5. This reflects the optimizer's efficiency in deploying the gradient for loss minimization and highlights the importance of the choice of optimizer.

Fig. 4.4 shows the distribution of optimized parameters of all successful runs for the Differometor optimization with AdamW and learning rate 0.1 at the full set of 31 optimized parameters. We observe that the optimized parameters don't match the default parameter values from code block 3.5. This shows that there is no unique solution to the optimization performed here and that the same setup topology with different parameters can lead to matching sensitivity curves.

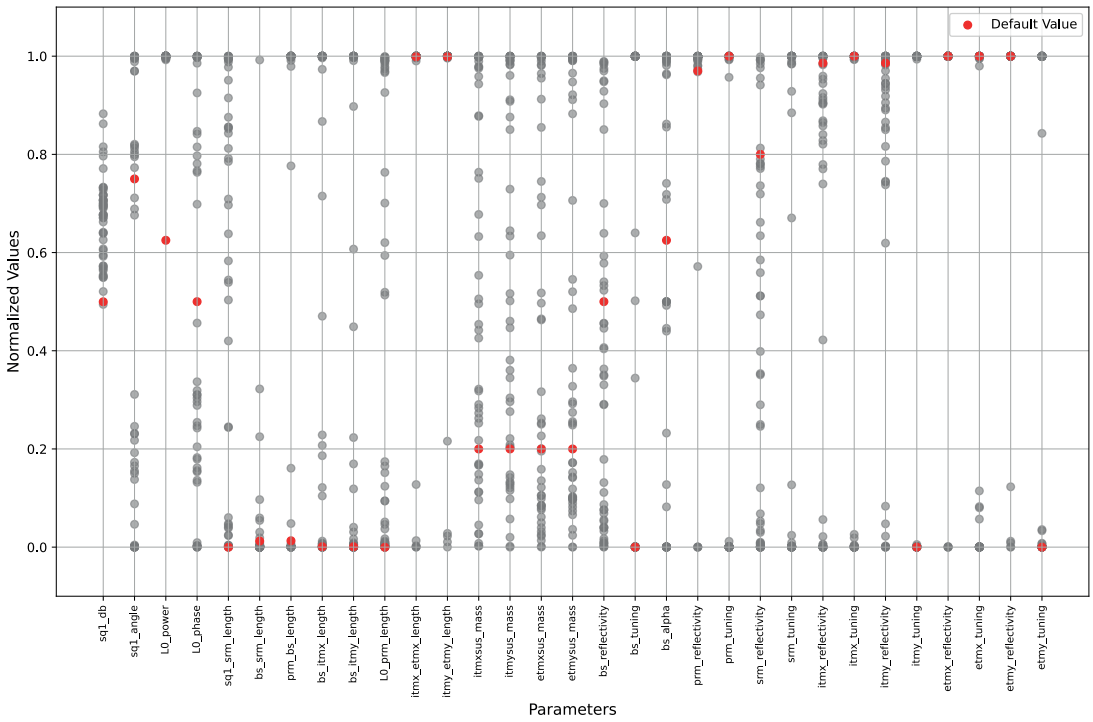


Figure 4.4: Parameter distributions for all successful optimizations. An optimization is successful if it crosses the loss threshold $10^{-5}$. Red dots indicate the default values of the simplified aLIGO setup from code block 3.5.

This experiment demonstrates the performance advantage that Differometor offers when running optimizations with large numbers of gradient evaluations. To find new gravitational wave detectors by optimizing larger UIFO setups as done by Krenn et al. [6] (see section 2.4.2), the optimizer needs to handle hundreds of parameters. In contrast to running Finesse with BFGS on CPU and with numerical gradient approximation, Differometor represents a promising alternative to tackle such larger optimizations.

## 4.2 Constrained Sensitivity Optimization of Advanced LIGO

The optimizations in section 4.1 are unconstrained which means that some of the resulting detector setups could result in light fields with high powers, potentially damaging optical components. Due to the implementation following the interferometer matrix approach, Differometor offers an easy way to prevent such high powers by applying additional constraints on all component port light fields. All these light fields are calculated by solving the system of equations defined by the interferometer matrix as shown in equation 2.28. To demonstrate this, we perform another experiment with the simplified aLIGO setup from Fig. 2.3 and code block 3.5. Instead of trying to match the sensitivity curve of the original setup, we now aim for a minimization of the sensitivity curves. Specifically, we change our loss function to the one described by equations 2.83 and 2.84. For the penalty function we choose a shifted rectified linear unit with a step defined by

$$f(p, co, c) = \begin{cases} 0 & \text{if } p \leq co \\ (p - co) + c & \text{if } p > co \end{cases}, \tag{4.2}$$

where $p$ is the power throughput, $co$ is the power cutoff value and $c$ defines the size of the penalty increase at this power cutoff. We choose the power cutoff values similar to the ones used by Krenn et al. [6], but adjust them so that our simplified aLIGO setup does not activate any penalties. Thus, we choose $co_h = 3.5$ MW as the power cutoff for the penalty function of reflecting objects, $co_s = 3$ kW for the penalty function of transmitting objects and $co_d = 10$ mW for the bleaching penalty at detectors. We identify transmitting and reflecting sides of an optical surface by calculating the light field with the highest power on each side and then defining the side with the higher power as the reflecting and the opposite side as the transmitting side. For all penalty function steps $c$ from equation 4.2 we choose 0.5 and for all the penalty weights $\alpha$, $\beta$, $\gamma$ from equation 2.83 we choose 0.25.

Based on an empirical hyperparameter exploration, we use the Adam optimizer [135] from Optax [132] together with gradient clipping with maximum global norm 1.0 [133], learning rate 0.1 and a learning rate schedule with multiple joined cosine decay cycles [136] as described in table 4.2.

| Phase | Init | Decay Steps | Warmup Steps | Peak | End |
|---|---|---|---|---|---|
| 1 | 0.1 | 2000 | 10 | 0.12 | 0.05 |
| 2 | 0.05 | 2000 | 10 | 0.06 | 0.025 |
| 3 | 0.025 | 2000 | 10 | 0.03 | 0.001 |
| 4 | 0.025 | 2000 | 10 | 0.0375 | 0.001 |
| 5 | 0.01 | 2000 | 10 | 0.012 | 0.0001 |
| 6 | 0.0001 | 1000 | 100 | 0.01 | 0.001 |
| 7 | 0.01 | 5000 | 100 | 0.1 | 0.0001 |

Table 4.2: Learning rate scheduler phases of multiple joined cosine decay cycles used for the constrained sensitivity optimization of aLIGO.

We use a similar procedure as for the rediscovery experiment and first generate 1000 setups where all optimized parameters are randomly initialized between $-10$ and 10. Before each simulation we again map all values to the range between 0 and 1 and then scale them to their respective physical ranges using the boundaries from table 4.1. For all 1000 setups, we calculate the loss for each one and start the optimization with the best performing one. We then run the optimization until they exceed 30.000 iterations or until the loss has not improved over 5000 iterations.

Fig. 4.5 A) shows the resulting losses for over 200 optimization runs. Around 39% of the runs reach losses below the loss of the simplified aLIGO setup at $-23.56$ without activating any penalty terms. The best performing runs reach losses as low as $-23.92$ with sensitivity curves that outperform the simplified aLIGO sensitivity curve over the entire frequency range. Fig. 4.5 B) and C) show the loss curves and resulting sensitivity curves of the five best performing optimizations. Again, the optimizations result in similar sensitivity curves while having different parameter distributions, indicating that there is no unique way to minimize the sensitivity curve.

This experiment demonstrates that Differometor supports optimizations with constraints on the light field power which is a crucial requirement for large scale discovery optimizations.

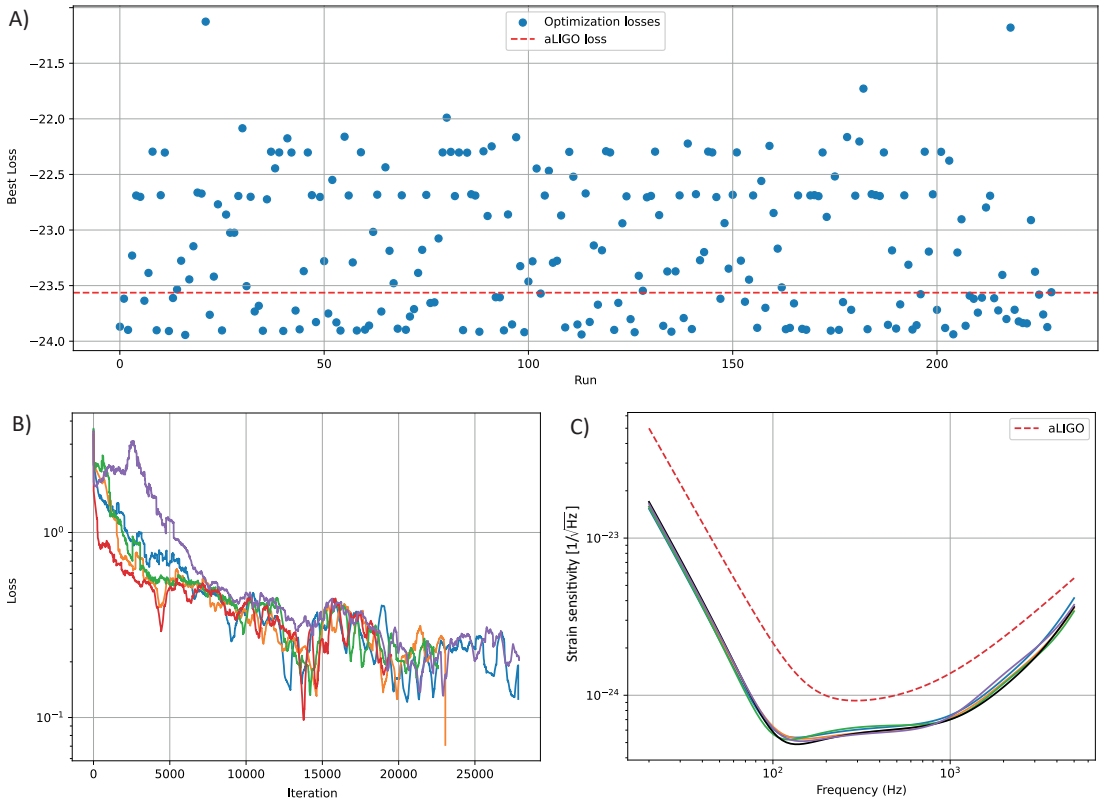


Figure 4.5: **A)** Best losses of all optimization runs. The red dashed line marks the loss of the simplified aLIGO setup from code block 3.5 that is used as a baseline. **B)** Smoothed loss curves of the five best performing optimizations. Loss curves have been shifted into the positive range by subtracting the smallest loss. **C)** Resulting sensitivity curves calculated from setups with the optimized parameters and the initial sensitivity curve from the simplified aLIGO setup from code block 3.5.

## 4.3 Towards an Automated Search for new Gravitational Wave Detectors

To demonstrate the scalability of Differometor to larger scale optimizations as done by Krenn et al. [6] for the digital discovery of new gravitational wave detectors, we implement the quasi-universal interferometer (UIFO) from Fig. 2.11 in Differometor and again run sensitivity optimizations of randomly initialized setups.

Specifically, we initialize UIFO setups of the sizes $n = 1$ to $n = 5$ where the resulting UIFO consists of a grid of $n \times n$ cells with either beam splitter or Faraday isolator. The

grid is surrounded by edge cells with either lasers, squeezers or detectors. Each UIFO is randomly initialized, meaning that components and orientations are chosen randomly. We then select all reflectivities, tunings, squeezing magnitudes, angles, powers, masses, lengths and phases as optimized parameters. Similar to the optimizations in sections 4.1 and 4.2, we calculate the loss for 1000 versions of each UIFO (100 versions for UIFOs of size 5 because of computational constraints), where the optimized parameters of each version are uniformly distributed between $-10$ and 10, get mapped to the range between 0 and 1 by applying a sigmoid function and then get scaled to their respective parameter ranges from table 4.1. We take the UIFO version with the best initial loss and optimize the parameters to minimize the loss from equation 2.83 with the same coefficients as in section 4.2. We again use the Adam optimizer [135] from Optax [132] with learning rate 0.1 or 0.01 and gradient clipping with maximum global norm 1.0 [133]. We don't use any learning rate scheduler and calculate the sensitivities on a frequency range from 20 Hz to 5000 Hz at 50 logarithmically-spaced values (35 values for UIFOs of size 5 because of computational constraints). We run optimizations until a maximum of 20.000 iterations or until there is no improvement in the loss over 5000 iterations.

Table 4.3 shows the number ranges of optimized parameters for each UIFO size. The exact number of parameters depends on the randomly initialized setup and can differ slightly. UIFOs of size 6 result in out of memory errors and therefore represent the current limit of Differometor optimizations when calculating the sensitivity at 50 frequency values. This can be further improved in future Differometor versions by e.g. switching to a sparse matrix representation.

| **UIFO Size** | **Number of optimized parameters** |
|---|---|
| 1 | 48 - 51 |
| 2 | 148 - 160 |
| 3 | 296 - 323 |
| 4 | 492 - 537 |
| 5 | 742 - 805 |

Table 4.3: Number ranges of optimized parameters for UIFOs of different sizes. The exact number of optimized parameters depends on the randomly initialized UIFO setup.

Fig. 4.6 A) shows the resulting best losses for all optimization runs at different UIFO sizes. No optimization of UIFOs of size 1 crossed the loss of the simplified aLIGO setup from code block 3.5. This is expected, as randomly initialized UIFOs of size 1 are mostly not capable of encoding the simplified aLIGO setup. For UIFOs of larger sizes, the best losses continually decrease below the simplified aLIGO baseline. Fig. 4.6 C) shows the strain sensitivity curve from the best performing optimization of UIFO size 4, which stopped after around 9 hours.

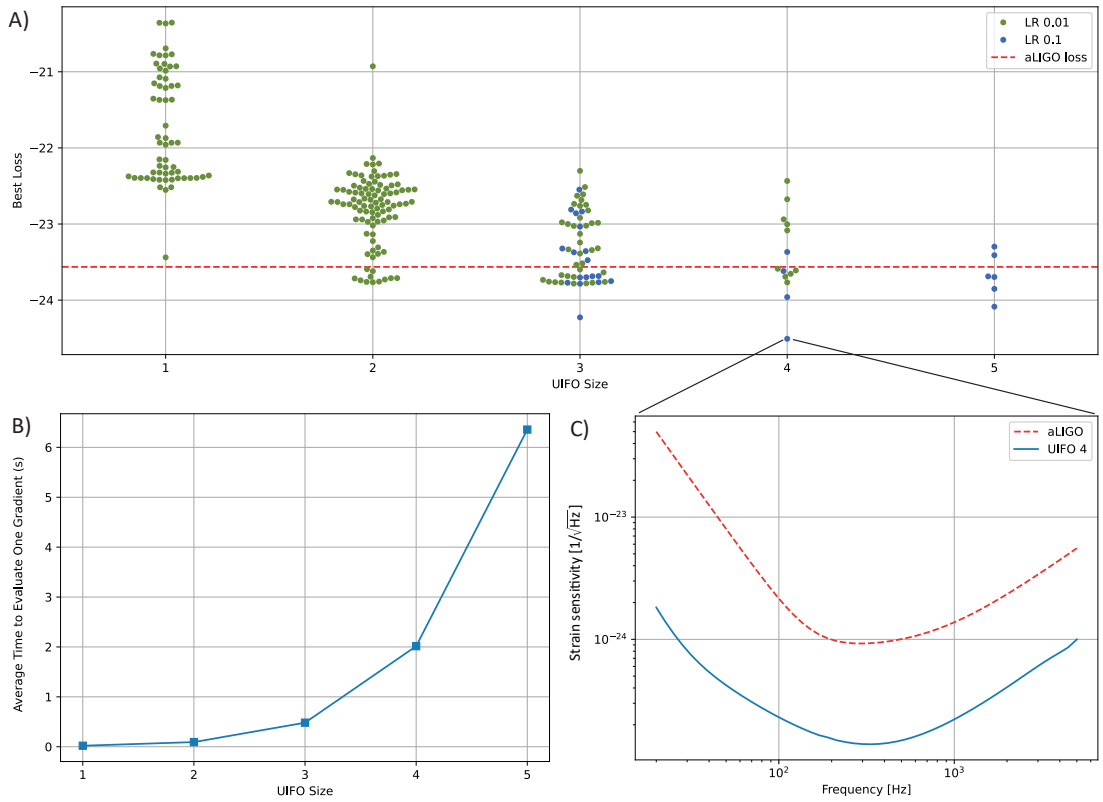


Figure 4.6: **A)** Best losses of all optimization runs at different UIFO sizes. The red dashed line marks the loss of the simplified aLIGO setup from code block 3.5 that is used as a baseline. **B)** Gradient evaluation timings (convergence time divided by the number of iterations) for optimizations with different UIFO sizes. **C)** The resulting strain sensitivity curve from the best performing optimization with UIFO size 4.

We again measure the average time to evaluate one gradient which we define as the convergence time divided by the number of iterations. Fig. 4.6 B) shows the results of this timing experiment.

The gradient evaluation time for Differometor and Adam passes 0.5 s at around 302 parameters, a value that Finesse and BFGS in Fig. 4.3 already reached at only 12 optimized parameters. However, the gradient evaluation time still strongly increases with the number of optimized parameters. Given that the interferometer matrix updates are the current computational bottleneck of Differometor, implementing a sparsification of the interferometer matrix again seems like a promising measure to improve this scaling behavior in future Differometor versions.

These results demonstrate the promising potential of future Differometor versions to take a key role in larger scale optimizations for the digital discovery of new gravitational wave detectors.

# Chapter 5

# Conclusions and Outlook

Differometor is a new differentiable frequency domain interferometer simulator, specifically designed for large-scale digital exploration of new superior quantum-enhanced hardware for fundamental physics research. The Differometor implementation closely follows the existing interferometer simulator FINESSE and offers carrier and signal propagation, quantum noise calculation and optomechanical effects as a large subset of the features of FINESSE. The choice of the JAX library provides significant advantages through GPU execution, just-in-time compilation and automatic differentiation, but also entails additional programming constraints like the restriction to pure functions and static inputs. Thus, Differometor adapts the FINESSE implementation towards functional and differentiable programming patterns by splitting all physics principles into single functions whose values are then distributed to the interferometer matrices via different indexing arrays.

Differometor is verified through comparison with FINESSE simulations, demonstrating good agreement of the calculated sensitivity curves with small discrepancies solely arising from differences in numerical accuracy. First performance tests show that Differometor simulations on GPUs outperform FINESSE simulations in terms of computation speed, demonstrating that Differometor makes good use of JAX parallelization and compilation features.

The promising potential of using Differometor for large-scale digital discovery of new gravitational wave detectors is demonstrated through different toy optimizations with a simplified version of the aLIGO detector. The experiments show that Differometor in combination with the Adam optimizer outperforms FINESSE used with BFGS in terms of convergence, gradient computation and convergence time, confirming that Differometor makes good use of JAX auto-differentiation features. The experiments further show that Differometor supports the formulation of additional constraints on the light fields within an optical setup which are necessary to enforce the feasibility of physical realization of optimized setups. Differometor optimizations are scalable to setups with hundreds of optimized parameters which is well suited for state-of-the-art digital discovery tasks.

Upcoming technical development plans include:

- Sparsification of interferometer matrices which is expected to further improve the performance and scalability,
- Improved handling of carrier, signal and sideband systems, which will enable Differometor to dynamically choose which systems are necessary for a specific simulation request,
- Several improvements in usability so that Differometor closely matches FINESSE in terms of setup definition and experienced users can easily switch without having to learn new standards,
- Implementation of several optimization features like coupling different parameters (e.g. lengths of parallel spaces in an UIFO) to the same optimized parameter or deploying ML-surrogates for setup initialization,
- Implementing new physics features to extend Differometor beyond the plane-wave approximation to new light field representations such as Gaussian beams which unlock the simulation of spatial effects through higher order modes.

Differometor is envisaged as a modular simulator that can be combined with simulators from other fields of physics like quantum optics or super-resolution microscopy to perform digital discovery on a search space that potentially includes novel fundamental physics experiments which lay outside of human reach through intuition and experience. Thus, Differometor is a step into the direction of AI for scientific understanding and can serve as a computational microscope and as a resource of inspiration for humans to come up with new ways to observe and better understand the universe.

# Appendix A

# Simplified aLIGO

| No. | Component | Type | Parameter | Optimized |
|---|---|---|---|---|
| 1 | sq1 | Squeezer | db | Yes |
| 2 | sq1 | Squeezer | angle | Yes |
| 3 | L0 | Laser | power | Yes |
| 4 | L0 | Laser | phase | Yes |
| 5 | sq1_srm | Space | length | Yes |
| 6 | bs_srm | Space | length | Yes |
| 7 | prm_bs | Space | length | Yes |
| 8 | bs_itmx | Space | length | Yes |
| 9 | bs_itmy | Space | length | Yes |
| 10 | L0_prm | Space | length | Yes |
| 11 | itmx_etmx | Space | length | Yes |
| 12 | itmy_etmy | Space | length | Yes |
| 13 | itmxsus | Free Mass | mass | Yes |
| 14 | itmysus | Free Mass | mass | Yes |
| 15 | etmxsus | Free Mass | mass | Yes |
| 16 | etmysus | Free Mass | mass | Yes |
| 17 | bs | Beamsplitter | reflectivity | Yes |
| 18 | bs | Beamsplitter | tuning | Yes |
| 19 | bs | Beamsplitter | alpha | Yes |
| 20 | prm | Mirror | reflectivity | Yes |
| 21 | prm | Mirror | tuning | Yes |
| 22 | srm | Mirror | reflectivity | Yes |
| 23 | srm | Mirror | tuning | Yes |
| 24 | itmx | Mirror | reflectivity | Yes |
| 25 | itmx | Mirror | tuning | Yes |
| 26 | itmy | Mirror | reflectivity | Yes |
| 27 | itmy | Mirror | tuning | Yes |

| No. | Component | Type | Parameter | Optimized |
|---|---|---|---|---|
| 28 | etmx | Mirror | reflectivity | Yes |
| 29 | etmx | Mirror | tuning | Yes |
| 30 | etmy | Mirror | reflectivity | Yes |
| 31 | etmy | Mirror | tuning | Yes |
| 32 | sq1_srm | Space | refractive_index | No |
| 33 | bs_srm | Space | refractive_index | No |
| 34 | prm_bs | Space | refractive_index | No |
| 35 | bs_itmx | Space | refractive_index | No |
| 36 | bs_itmy | Space | refractive_index | No |
| 37 | L0_prm | Space | refractive_index | No |
| 38 | itmx_etmx | Space | refractive_index | No |
| 39 | itmy_etmy | Space | refractive_index | No |
| 40 | bs | Beamsplitter | loss | No |
| 41 | prm | Mirror | loss | No |
| 42 | srm | Mirror | loss | No |
| 43 | itmx | Mirror | loss | No |
| 44 | itmy | Mirror | loss | No |
| 45 | etmx | Mirror | loss | No |
| 46 | etmy | Mirror | loss | No |

Table A.1: Parameters of the simplified aLIGO setup from Fig. 2.3 and code block 3.5.

# Bibliography


[1] J. Aasi, B. P. Abbott, R. Abbott, T. Abbott, M. R. Abernathy, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. Adhikari, and et al., Advanced ligo, Classical and Quantum Gravity **32**, 074001 (2015).

[2] B. P. Abbott, R. Abbott, T. Abbott, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. Adhikari, V. B. Adya, and et al., Multi-messenger observations of a binary neutron star merger, The Astrophysical Journal Letters **848**, L12 (2017).

[3] S. L. Danilishin and F. Y. Khalili, Quantum measurement theory in gravitational-wave detectors, Living Reviews in Relativity **15** (2012).

[4] M. Krenn, M. Erhard, and A. Zeilinger, Computer-inspired quantum experiments, Nature Reviews Physics **2**, 649 – 661 (2020).

[5] M. Krenn, M. Malik, R. Fickler, R. Lapkiewicz, and A. Zeilinger, Automated search for new quantum experiments., Physical review letters **116 9**, 090405 (2015).

[6] M. Krenn, Y. Drori, and R. X. Adhikari, Digital discovery of interferometric gravitational wave detectors, arXiv preprint (2023).

[7] A. Freise, G. Heinzel, H. Lück, R. Schilling, B. Willke, and K. Danzmann, Frequency-domain interferometer simulation with higher-order spatial modes, Classical and Quantum Gravity **21**, S1067 – S1074 (2003).

[8] A. Freise, D. D. Brown, and C. Bond, Finesse 2 manual (2014), accessed: 2025-01-28.

[9] D. D. Brown, A. Freise, and et al., Finesse 3 manual (2017), accessed: 2025-01-28.

[10] C. R. Harris, K. J. Millman, S. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, and et al., Array programming with numpy, Nature **585**, 357 – 362 (2020).

[11] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, JAX: composable transformations of Python+NumPy programs (2018).

[12] A. Einstein, Näherungsweise integration der feldgleichungen der gravitation, Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften , 688–696 (1916).

[13] R. A. Hulse and J. H. Taylor, Discovery of a pulsar in a binary system, The Astrophysical Journal **195**, L51–L53 (1975).

[14] J. H. Taylor and J. M. Weisberg, A new test of general relativity - Gravitational radiation and the binary pulsar PSR 1913+16, The Astrophysical Journal **253**, 908–920 (1982).

[15] B. P. Abbott, R. Abbott, T. D. Abbott, M. R. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. Adhikari, and et al. (LIGO Scientific Collaboration and Virgo Collaboration), Observation of gravitational waves from a binary black hole merger, Phys. Rev. Lett. **116**, 061102 (2016).

[16] P. Saulson, *Fundamentals Of Interferometric Gravitational Wave Detectors* (World Scientific Publishing Company, 1994).

[17] M. Bailes, B. K. Berger, P. R. Brady, M. Branchesi, K. Danzmann, M. J. Evans, K. Holley-Bockelmann, B. R. Iyer, T. Kajita, S. Katsanevas, and et al., Gravitational-wave physics and astronomy in the 2020s and 2030s, Nature Reviews Physics **3**, 344 – 366 (2021).

[18] T. Hartwig, M. Volonteri, V. Bromm, R. S. Klessen, E. Barausse, M. Magg, and A. Stacy, Gravitational waves from the remnants of the first stars, Mon. Not. Roy. Astron. Soc. **460**, L74–L78 (2016).

[19] S. Bernuzzi, T. Dietrich, and A. Nagar, Modeling the complete gravitational wave spectrum of neutron star mergers., Physical review letters **115 9**, 091101 (2015).

[20] C. D. Ott, The gravitational-wave signature of core-collapse supernovae, Classical and Quantum Gravity **26**, 063001 (2008).

[21] D. Vartanyan, A. Burrows, T. Wang, M. S. B. Coleman, and C. J. White, Gravitational-wave signature of core-collapse supernovae, Phys. Rev. D **107**, 103015 (2023).

[22] T. Zhang, H. Yang, D. Martynov, P. Schmidt, and H. Miao, Gravitational-wave detector for postmerger neutron stars: Beyond the quantum loss limit of the fabry-perot-michelson interferometer, Phys. Rev. X **13**, 021019 (2022).

[23] R. D. Blandford and K. S. Thorne, Applications of classical physics, Lecture Notes, California Institute of Technology (2012).

[24] J. Mizuno, Comparison of optical configurations for laser-interferometric gravitational-wave detectors, Ph.D. thesis, Hannover Universität (1995).

[25] C. Z. Bond, D. D. Brown, A. Freise, and K. Strain, Interferometer techniques for gravitational-wave detection, Living Reviews in Relativity **19** (2017).

[26] A. Buikema, C. Cahillane, G. L. Mansell, C. D. Blair, R. Abbott, C. Adams, R. X. Adhikari, A. Ananyeva, S. Appert, K. Arai, and et al., Sensitivity and performance of the advanced ligo detectors in the third observing run, Phys. Rev. D **102**, 062003 (2020).

[27] V. B. Braginskii and Y. I. Vorontsov, Quantum-mechanical limitations in macroscopic experiments and modern experimental technique, Physics-Uspekhi **17**, 644–650 (1975).

[28] M.-T. Jaekel and S. Reynaud, Quantum limits in interferometric measurements, EPL **13**, 301–306 (1990).

[29] Meers and Strain, Modulation, signal, and quantum noise in interferometers., Physical review. A, Atomic, molecular, and optical physics **44 7**, 4693–4703 (1991).

[30] Niebauer, Schilling, Danzmann, Rüdiger, and Winkler, Nonstationary shot noise and its effect on the sensitivity of interferometers., Physical review. A, Atomic, molecular, and optical physics **43 9**, 5022–5029 (1991).

[31] H. J. Kimble, Y. Levin, A. B. Matsko, K. S. Thorne, and S. P. Vyatchanin, Conversion of conventional gravitational-wave interferometers into quantum nondemolition interferometers by modifying their input and/or output optics, Physical Review D **65**, 022002 (2000).

[32] M. Ando, K. Arai, R. Takahashi, G. Heinzel, S. Kawamura, D. Tatsumi, N. Kanda, H. Tagoshi, A. Araya, H. Asada, and et al. (Tama collaboration), Stable operation of a 300-m laser interferometer with sufficient sensitivity to detect gravitational-wave events within our galaxy, Physical review letters **86 18**, 3950–4 (2001).

[33] B. Willke, P. Aufmuth, C. Aulbert, S. Babak, R. Balasubramanian, B. W. Barr, S. Berukoff, S. Bose, G. Cagnoli, M. M. Casey, and et al., The geo 600 gravitational wave detector, Classical and Quantum Gravity **19**, 1377–1387 (2002).

[34] T. Accadia, F. Acernese, M. Alshourbagy, P. Amico, F. Antonucci, S. Aoudia, N. Arnaud, C. Arnault, K. G. Arun, P. Astone, and et al., Virgo: a laser interferometer to detect gravitational waves, Journal of Instrumentation **7** (03), P03012.

[35] D. Sigg and et al., Status of the ligo detectors, Classical and Quantum Gravity **25**, 114041 (2006).

[36] F. Acernese, M. Agathos, K. Agatsuma, D. Aisa, N. Allemandou, A. Allocca, J. Amarni, P. Astone, G. Balestri, G. Ballardin, and et al., Advanced virgo: a second-generation interferometric gravitational wave detector, Classical and Quantum Gravity **32** (2014).

[37] M. Evans, R. X. Adhikari, C. Afle, S. W. Ballmer, S. Biscoveanu, S. Borhanian, D. A. Brown, A. Duncan, Y. Chen, and R. E. et al., A horizon study for cosmic explorer: Science, observatories, and community, arXiv preprint (2021).

[38] D. Reitze, R. X. Adhikari, S. Ballmer, B. Barish, L. Barsotti, G. Billingsley, D. A. Brown, Y. Chen, and et al., Cosmic Explorer: The U.S. Contribution to Gravitational-Wave Astronomy beyond LIGO, arXiv preprint (2019).

[39] M. Punturo, M. Abernathy, F. Acernese, B. Allen, N. Andersson, K. Arun, F. Barone, B. Barr, M. Barsuglia, M. Beker, and et al., The Einstein Telescope: A third-generation gravitational wave observatory, Class. Quant. Grav. **27**, 194002 (2010).

[40] E. D. Hall and M. J. Evans, Metrics for next-generation gravitational-wave detectors, Classical and Quantum Gravity **36** (2019).

[41] K. Danzmann and the LISA study team, Lisa: laser interferometer space antenna for gravitational wave measurements, Classical and Quantum Gravity **13**, A247 (1996).

[42] M. Abe, P. Adamson, M. Borcean, D. Bortoletto, K. Bridges, and S. P. C. et al., Matter-wave atomic gradiometer interferometric sensor (magis-100), Quantum Science and Technology **6**, 044003 (2021).

[43] S. W. Ballmer and et al., Snowmass2021 cosmic frontier white paper: Future gravitational-wave detector facilities, arXiv preprint (2022).

[44] C. Cahillane and G. L. Mansell, Review of the advanced ligo gravitational wave observatories leading to observing run four, Galaxies (2022).

[45] C. Z. Bond, D. D. Brown, and A. Freise, Interferometer responses to gravitational waves: Comparing finesse simulations and analytical solutions, arXiv: preprint (2013).

[46] H. Billing, K. Maischberger, A. Rüdiger, R. Schilling, L. Schnupp, and W. Winkler, The munich gravitational wave detector using laser interferometry, *Quantum Optics, Experimental Gravity, and Measurement Theory*, NATO ASI Series B, **94**, 525–566 (1983).

[47] R. Drever, J. Hough, A. Munley, S. Lee, R. Spero, S. Whitcomb, and et al., Gravitational wave detectors using laser interferometers and optical cavities: ideas, principles and prospects, *Quantum Optics, Experimental Gravity, and Measurement Theory*, NATO ASI Series B, **94**, 503–514 (1983).

[48] B. J. Meers, Recycling in laser-interferometric gravitational-wave detectors., Physical review. D, Particles and fields **38 8**, 2317–2326 (1988).

[49] W. Nolting, Grundkurs theoretische physik 3 - elektrodynamik, Springer Lehrbuch (2011).

[50] J. Mizuno and I. Yamaguchi, Method for analyzing multiple-mirror coupled optical systems, Journal of The Optical Society of America A-optics Image Science and Vision **16**, 1730–1739 (1999).

[51] G. Heinzel, Advanced optical techniques for laser-interferometric gravitational-wave detectors, PhD Thesis, Hannover Universität (1999).

[52] C. M. Caves, Quantum mechanical noise in an interferometer, Physical Review D **23**, 1693–1708 (1981).

[53] C. M. Caves and B. L. Schumaker, New formalism for two-photon quantum optics. i. quadrature phases and squeezed states., Physical review. A, General physics **31 5**, 3068–3092 (1985).

[54] R. Paschotta, Noise of mode-locked lasers (part i): numerical model, Applied Physics B **79**, 153–162 (2004).

[55] R. Paschotta, Noise of mode-locked lasers (part ii): timing jitter and other fluctuations, Applied Physics B **79**, 163–173 (2004).

[56] R. Paschotta, A. Schlatter, S. C. Zeller, H. R. Telle, and U. Keller, Optical phase noise and carrier-envelope offset noise of mode-locked lasers, Applied Physics B **82**, 265–273 (2006).

[57] H. B. Callen and T. A. Welton, Irreversibility and generalized noise, Physical Review **83**, 34–40 (1951).

[58] L. D. Landau and E. M. Lifshitz, Statistical Physics, Part 1, Butterworth-Heinemann **5** (1980).

[59] D. D. Brown, Interactions of light and mirrors: advanced techniques for modelling future gravitational wave detectors, PhD Thesis, University of Birmingham (2016).

[60] J. Harms, P. Cochrane, and A. Freise, Quantum-noise power spectrum of fields with discrete classical components, Physical Review A **76**, 023803 (2007).

[61] D. N. Klyshko, Coherent Photon Decay in a Nonlinear Medium, ZhETF Pisma Redaktsiiu **6**, 490 (1967).

[62] D. Walls and G. Milburn, Quantum optics, Springer Berlin Heidelberg (2008).

[63] P. Meystre, E. M. Wright, J. D. McCullen, and E. Vignes, Theory of radiation-pressure-driven interferometers, Journal of The Optical Society of America B-optical Physics **2**, 1830–1840 (1985).

[64] O. Miyakawa and H. Yamamoto, Lock acquisition studies for advanced interferometers, Journal of Physics: Conference Series **122**, 012024 (2008).

[65] M. J. Evans, N. Mavalvala, P. Fritschel, R. Bork, B. Bhawal, R. Gustafson, W. P. Kells, M. Landry, D. Sigg, R. Weiss, S. E. Whitcomb, and H. Yamamoto, Lock acquisition of a gravitational-wave interferometer., Optics letters **27 8**, 598–600 (2002).

[66] V. B. Braginsky, S. E. Strigin, and S. P. Vyatchanin, Parametric oscillatory instability in fabry-perot interferometer, Physics Letters A **287**, 331–338 (2001).

[67] A. Buonanno and Y. Chen, Signal recycled laser-interferometer gravitational-wave detectors as optical springs, Physical Review D **65**, 042001 (2001).

[68] B. S. Sheard, M. B. Gray, C. M. Mow-Lowry, D. McClelland, and S. E. Whitcomb, Observation and characterization of an optical spring, Physical Review A **69**, 051801 (2004).

[69] M. Aspelmeyer, T. J. Kippenberg, and F. Marquardt, Cavity optomechanics, Rev. Mod. Phys. **86**, 1391–1452 (2014).

[70] J.-Y. Vinet, P. Hello, C. N. Man, and A. Brillet, A high accuracy method for the simulation of non-ideal optical cavities, Journal De Physique I **2**, 1287–1303 (1992).

[71] B. Bhawal, G. Cella, M. Evans, S. Klimenko, E. Maros, S. D. Mohanty, M. Rakhmanov, R. L. Savage, and H. Yamamoto, LIGO end-to-end simulation program, American Institute of Physics Conference Series **523**, 469–470 (2000).

[72] B. Caron, L. Derome, R. Flaminio, X. Grave, F. Marion, B. Mours, D. Verkindt, F. Cavalier, and A. Viceré, Siesta, a time domain, general purpose simulation program for the virgo experiment, Astroparticle Physics **10**, 369–386 (1999).

[73] H. Yamamoto, M. Barton, B. Bhawal, M. Evans, and S. Yoshida, Simulation tools for future interferometers, Journal of Physics: Conference Series **32**, 398 (2006).

[74] H. Yamamoto, Simulations as crucial tools for gw experiments, IEEE Symposium Conference Record Nuclear Science 2004. **7**, 4618–4624 Vol. 7 (2004).

[75] B. Bhawal, *The Effect of Thermal Lensing on Wave-Front Sensor signals*, Tech. Rep. T040066-E (LIGO Document, 2004).

[76] H. Yamamoto, *Recent LIGO I Simulation Results*, Tech. Rep. G030417-E (LIGO Document, 2003).

[77] M. Barton, *Models of the Advanced LIGO Suspensions in Mathematica*, Tech. Rep. T020205-D (LIGO Document, 2002).

[78] B. Lantz, *Snapshot of the SEI Model for e2e*, Tech. Rep. T050141-R (LIGO Document, 2005).

[79] B. Bochner and Y. Hefetz, Grid-based simulation program for gravitational wave interferometers with realistically imperfect optics, Physical Review D **68**, 082001 (2003).

[80] H. Yamamoto, *FFT Study of Mode Matching at LHO and LLO*, Tech. Rep. G050229 (LIGO Document, 2005).

[81] J. Degallaix, Oscar a matlab based optical fft code, Journal of Physics: Conference Series **228**, 012021 (2010).

[82] J. Degallaix, Oscar: A matlab based package to simulate realistic optical cavities, SoftwareX **12**, 100587 (2020).

[83] H. Yamamoto, Sis (stationary interferometer simulation) manual (2013), accessed: 2025-01-28.

[84] M. Evans and R. L. Ward, Optickle (2015), accessed: 2025-01-28.

[85] G. Vajente, Fast modal simulation of paraxial optical systems: the mist open source toolbox, Classical and Quantum Gravity **30** (2013).

[86] C. Z. Bond, How to stay in shape: overcoming beam and mirror distortions in advanced gravitational wave interferometers, PhD Thesis, University of Birmingham (2014).

[87] G. Vajente, Full modal simulation of opto-mechanical effects in optical systems, Classical and Quantum Gravity **31**, 075005 (2014).

[88] J. W. Richardson, S. Pandey, E. Bytyqi, T. B. Edo, and R. X. Adhikari, Optimizing gravitational-wave detector design for squeezed light, Physical Review D (2022).

[89] S. Ballmer, J. Degallaix, A. Freise, and P. Fulda, *Comparing Finesse simulations, analytical solutions and OSCAR simulations of Fabry-Perot alignment signals*, Tech. Rep. (LIGO, 2013) lIGO note T1300345.

[90] C. Bond, P. Fulda, D. Brown, and A. Freise, *Comparing Simulations of the Advanced LIGO Dual-Recycled Michelson*, Tech. Rep. (LIGO, 2013) lIGO note T1400270.

[91] M. Krenn, R. Pollice, S. Y. Guo, M. Aldeghi, A. Cervera-Lierta, P. Friederich, G. dos Passos Gomes, F. Hase, A. Jinich, A. Nigam, Z. Yao, and A. Aspuru-Guzik, On scientific understanding with artificial intelligence, Nature Reviews. Physics **4**, 761 – 769 (2022).

[92] H. Wang, T. Fu, Y. Du, W. Gao, K. Huang, Z. Liu, P. Chandak, S. Liu, P. V. Katwyk, A. Deac, and et al., Scientific discovery in the age of artificial intelligence, Nature **620**, 47–60 (2023).

[93] G. Liu, D. B. Catacutan, K. Rathod, K. Swanson, W. Jin, J. C. Mohammed, A. Chiappino-Pepe, S. Syed, M. Fragis, K. Rachwalski, and et al., Deep learning-guided discovery of an antibiotic targeting acinetobacter baumannii, Nature Chemical Biology **19**, 1342–1350 (2023).

[94] R. Gómez-Bombarelli, J. Aguilera-Iparraguirre, T. D. Hirzel, D. K. Duvenaud, D. Maclaurin, and et al., Design of efficient molecular organic light-emitting diodes by a high-throughput virtual screening and experimental approach., Nature materials **15 10**, 1120–7 (2016).

[95] A. A. Sadybekov, A. V. Sadybekov, Y. Liu, C. Iliopoulos-Tsoutsouvas, X.-P. Huang, J. E. Pickett, B. Houser, N. Patel, and et al., Synthon-based ligand discovery in virtual libraries of over 11 billion compounds, Nature **601**, 452 – 459 (2021).

[96] J. M. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, and et al., Highly accurate protein structure prediction with alphafold, Nature **596**, 583 – 589 (2021).

[97] A. Zhavoronkov, Y. A. Ivanenkov, A. Aliper, M. Veselov, V. Aladinskiy, A. V. Aladinskaya, V. Terentiev, and et al., Deep learning enables rapid identification of potent ddr1 kinase inhibitors, Nature Biotechnology **37**, 1038 – 1040 (2019).

[98] Y. Bengio, S. Lahlou, T. Deleu, E. J. Hu, M. Tiwari, and E. Bengio, Gflownet foundations, J. Mach. Learn. Res. **24** (2023).

[99] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, Grammar variational autoencoder, *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, 1945–1954 (2017).

[100] R. Gómez-Bombarelli, D. K. Duvenaud, J. M. Hernández-Lobato, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, Automatic chemical

design using a data-driven continuous representation of molecules, ACS Central Science **4**, 268 – 276 (2016).

[101] M. H. S. Segler, M. Preuss, and M. P. Waller, Planning chemical syntheses with deep neural networks and symbolic ai, Nature **555**, 604–610 (2017).

[102] W. Gao, P. Raghavan, and C. W. Coley, Autonomous platforms for data-driven organic synthesis, Nature Communications **13** (2022).

[103] J. Degrave, F. Felici, J. Buchli, M. Neunert, and B. D. T. et al., Magnetic control of tokamak plasmas through deep reinforcement learning, Nature **602**, 414 – 419 (2022).

[104] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang, Autonomous navigation of stratospheric balloons using reinforcement learning, Nature **588**, 77 – 82 (2020).

[105] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, Physics-informed machine learning, Nature Reviews Physics **3**, 422 – 440 (2021).

[106] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar, Physics-informed neural operator for learning partial differential equations, ACM / IMS J. Data Sci. **1** (2024).

[107] K. Cranmer, J. Brehmer, and G. Louppe, The frontier of simulation-based inference, Proceedings of the National Academy of Sciences **117**, 30055 – 30062 (2019).

[108] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles, Variational quantum algorithms, Nature Reviews Physics **3**, 625 – 644 (2020).

[109] M. Ostaszewski, L. M. Trenkwalder, W. Masarczyk, E. Scerri, and V. Dunjko, Reinforcement learning for optimization of variational quantum circuit architectures, Advances in Neural Information Processing Systems **34**, 18182–18194 (2021).

[110] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke, *et al.*, Noisy intermediate-scale quantum algorithms, Reviews of Modern Physics **94**, 015004 (2022).

[111] S. Molesky, Z. Lin, A. Piggott, W. Jin, J. Vuković, and A. W. Rodriguez, Inverse design in nanophotonics, Nature Photonics **12**, 659 – 670 (2018).

[112] W. Ma, Z. Liu, Z. A. Kudyshev, A. Boltasseva, W. Cai, and Y. Liu, Deep learning for the design of photonic structures, Nature Photonics **15**, 77 – 90 (2020).

[113] C. Rodríguez, S. Arlt, L. Mockl, and M. Krenn, Automated discovery of experimental designs in super-resolution microscopy with xlumina, Nature Communications **15** (2024).

[114] C. G. Broyden, The convergence of a class of double-rank minimization algorithms 2. the new algorithm, Ima Journal of Applied Mathematics **6**, 222–231 (1970).

[115] R. Fletcher, A new approach to variable metric algorithms, Comput. J. **13**, 317–322 (1970).

[116] D. Goldfarb, A family of variable-metric methods derived by variational means, Mathematics of Computation **24**, 23–26 (1970).

[117] D. F. Shanno, Conditioning of quasi-newton methods for function minimization, Mathematics of Computation **24**, 647–656 (1970).

[118] J. Aycock, A brief history of just-in-time, ACM Comput. Surv. **35**, 97–113 (2003).

[119] A. G. Baydin, B. A. Pearlmutter, A. Radul, and J. M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. **18**, 153:1–153:43 (2015).

[120] C. C. Margossian, A review of automatic differentiation and its efficient implementation, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **9** (2018).

[121] M. Blondel and V. Roulet, The elements of differentiable programming, arXiv preprint (2024).

[122] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, Difftaichi: Differentiable programming for physical simulation, ICLR (2020).

[123] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, Julia: A fresh approach to numerical computing, SIAM Review **59**, 65–98 (2017).

[124] M. AlQuraishi, End-to-end differentiable learning of protein structure, Cell Systems **8**, 292–301.e3 (2019).

[125] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, Brax - a differentiable physics engine for large scale rigid body simulation (2021).

[126] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer, Machine learning–accelerated computational fluid dynamics, Proceedings of the National Academy of Sciences of the United States of America **118** (2021).

[127] S. S. Schoenholz and E. D. Cubuk, Jax, m.d. a framework for differentiable physics, Journal of Statistical Mechanics: Theory and Experiment **2021** (2020).

[128] L. Heinrich and M. Kagan, Differentiable matrix elements with madjax, Journal of Physics: Conference Series **2438**, 012137 (2023).

[129] M. Innes, A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V. B. Shah, and W. Tebbutt, A differentiable programming system to bridge machine learning and scientific computing, arXiv preprint (2019).

[130] A. A. Hagberg, D. A. Schult, P. Swart, and J. Hagberg, Exploring network structure, dynamics, and function using networkx, Proceedings of the Python in Science Conference (2008).

[131] I. Loshchilov and F. Hutter, Decoupled weight decay regularization, arXiv preprint (2017).

[132] DeepMind, I. Babuschkin, K. Baumli, A. Bell, S. Bhupatiraju, J. Bruce, P. Buchlovsky, D. Budden, T. Cai, A. Clark, and et al., The DeepMind JAX Ecosystem (2020).

[133] R. Pascanu, T. Mikolov, and Y. Bengio, On the difficulty of training recurrent neural networks, Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28 (2013).

[134] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, and et al., SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods **17**, 261–272 (2020).

[135] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, Proceedings of 3rd International Conference on Learning Representations (2015).

[136] I. Loshchilov and F. Hutter, Sgdr: Stochastic gradient descent with warm restarts, arXiv preprint (2016).