Bachelorarbeit

Institut für Theoretische Physik
Computational Physics

Gradient-based optimization of AMEA parameters

Advisor:
Univ.-Prof. Dipl.-Phys. Dr.rer.nat. Wolfgang von der Linden

Franz Scherr

Mat.Nr. 01331085

Sommersemester 2017

Contents

1 Motivation **3**

2 Introduction **3**
2.1 Open Quantum Systems . . . 3
2.2 Auxiliary Systems - AMEA . . . 4
2.3 Gradient-based Optimization . . . 5
2.4 ADAM . . . 5

3 Parametrization **7**
3.1 Symmetric Problems . . . 7
3.1.1 E Matrix . . . 7
3.1.2 Γ Matrices . . . 8
3.1.3 H Full Matrix Parametrization . . . 9
3.1.4 H Variable-Rank Parametrization . . . 9
3.1.5 Relationship of the two Γ matrices . . . 10
3.2 General, Non-symmetric Problems . . . 10

4 Derivation of the Gradient **10**
4.1 Derivative of G^R_{ff} . . . 11
4.2 Derivative of G^K_{ff} . . . 12
4.3 Derivative of E . . . 12
4.4 Derivative of Γ^α . . . 13
4.5 Full Gradient . . . 13

5 Implementation **13**
5.1 Numerical Integration . . . 13
5.2 Matrix Inversion . . . 14
5.3 Sanity Check with Finite Differences . . . 15
5.4 Tensorflow . . . 15

6 Results **17**
6.1 Reference Physical System . . . 17
6.2 Reference Solution - Parallel Tempering . . . 17
6.3 Reference Solution - Nonimprovable . . . 19
6.4 Optimizing with Full Matrix Parametrization . . . 20
6.5 Optimizing with Variable Rank Parametrization . . . 23
6.5.1 Dependence on the Rank M . . . 25
6.6 Runtime Comparison . . . 27

7 Conclusions **27**

Abstract

Auxiliary Master Equation Approach (AMEA) [Dorda et al., 2014] is a generic framework to deal with quantum impurity problems. Numerical approaches to fit the corresponding parameters were by now based on parallel tempering algorithms to find the global optima that describe a physical system as good as possible. In this document gradient-based approaches are considered with different ways of parametrization. As a new result, the gradient-based approach, with iteratively increasing number of parameters, allows to find a comparable good cost function minimum while maintaining the performance of typical gradient-based algorithms.

1 Motivation

The theory of open quantum systems aims to describe interactions between quantum systems, one of which is typically a large environment. Such systems are not only an interesting realm of quantum physics on its own but also, out of this theory, new technology is emerging that has the potential to greatly enhance many areas in life. A prominent example are quantum dots that endow LCD TVs with better backlighting. One way to analyze such systems is to find alternative, auxiliary descriptions that are more accessible for further investigation like the Auxiliary Master Equation Approach. However to use this framework, characteristic parameters are required. In this work, a gradient-based approach to find the values of these parameters is established.

2 Introduction

2.1 Open Quantum Systems

A quantum system is usually described by a Hamiltonian as it is the generator of the so-called time evolution operator, a unitary operator. However, this approach makes the assumption that the system is closed and no interaction with an environment can occur. Thus, if the considered quantum system interacts with the environment, the environment has to be included into the hamiltonian. The problem of which is that solving this eigenvalue problem of the full system is cumbersome and the interesting properties or observables are only influenced by the part of the state that corresponds to the considered system. That is, we are only interested in the density operator of the considered system, tracing out the environment.

Assume that the considered system is denoted with A and the environment is labelled with B, furthermore let $\hat{\rho}$ denote the density operator of the entire system $A \otimes B$. Then we are interested in the time evolution of the reduced density operator of the considered system:

$$\hat{\rho}_A(t) = \mathrm{tr}_B(\hat{\rho}(t))$$

Given some requirements, according to [Breuer and Petruccione, 2007], this can also be represented with the time evolution of the reduced system itself plus additional terms

that describe the interaction of environment and system. This general form is called **Lindblad** equation and is a markovian master equation. It has the form:

$$\frac{\partial}{\partial t}\hat{\rho}_A = -\frac{i}{\hbar}\left[\hat{H}_A, \hat{\rho}_A\right] + \gamma \sum_j \left[\hat{L}_j \hat{\rho}_A \hat{L}_j^\dagger - \frac{1}{2}\left\{\hat{L}_j^\dagger \hat{L}_j, \hat{\rho}_A\right\}\right],$$

where the total Hamiltonian is split into the systems hamiltonians plus an interaction term $\hat{H} = \hat{H}_A + \hat{H}_B + \hat{H}_{AB}$ and the $\hat{L}_j$ denote the so-called Lindblad operators. By the structure of this equation one is able to identify that the first term describes the systems time evolution without environmental influence and the second term accounts for interaction. In addition, it is important to note that the second term is not a unitary operator. Thus, it can account for dissipation and decoherence.

2.2 Auxiliary Systems - AMEA

From the general **Lindblad** equation the **Auxiliary Master Equation Approach** emerged with the use of Green's functions and **Nonequilibrium Dynamical Mean-Field Theory** [Arrigoni et al., 2013], [Dorda et al., 2014].

Subject to this work is not to consider the theory behind open quantum systems but approximations to such systems and hence, we will focus on the parts that connect approximation and real system. We are interested in so-called impurity problems that are defined by the systems hamiltonian and a bath hybridization function. This function is typically given and the goal is to find an auxiliary system in which the systems hamiltonian is exposed to the same hybridization function. The auxiliary system is then composed of several bath sites that try to approximate the original system. A certain bath site corresponds to the impurity. In the limit of an infinite number of bath sites, the approximation becomes exact and by increasing the number of bath sites the approximation gets exponentially better.

Consider the auxiliary systems Green's function $\boldsymbol{G}(\omega)$. This total Green's function is composed of the retarded and Keldysh components, denoted by $\boldsymbol{G}^R(\omega)$ and $\boldsymbol{G}^K(\omega)$ respectively which are matrices of size $(n_b + 1) \times (n_b + 1)$, if n_b denotes the number of auxiliary bath sites. Furthermore, let G^α_{ff} ($\alpha \in \{R, K\}$) denote the matrix element that corresponds to the impurity, i.e. ff is the index of the impurity. In addition there is the Green's function of the isolated impurity denoted by $g^R(\omega)$. The corresponding hybridization functions of the auxiliary system are referred to as $\Delta^R_{\text{aux}}(\omega), \Delta^K_{\text{aux}}(\omega)$ for retarded and Keldysh components respectively. We then have for the auxiliary system:

$$\begin{aligned}
\boldsymbol{G}^R(\omega) &= (\omega - \boldsymbol{E} + i(\boldsymbol{\Gamma}^{(1)} + \boldsymbol{\Gamma}^{(2)}))^{-1}, \\
\boldsymbol{G}^K(\omega) &= 2i\boldsymbol{G}^R(\omega)(\boldsymbol{\Gamma}^{(2)} - \boldsymbol{\Gamma}^{(1)})\boldsymbol{G}^A(\omega)
\end{aligned}$$

$$\begin{aligned}
\Delta^R_{\text{aux}}(\omega) &= \frac{1}{g^R(\omega)} - \frac{1}{G^R_{ff}(\omega)}, \\
\Delta^K_{\text{aux}}(\omega) &= \frac{G^K_{ff}(\omega)}{|G^R_{ff}(\omega)|^2}
\end{aligned}$$

The considered problem is to find values for the parameters $\boldsymbol{E}, \boldsymbol{\Gamma}^{(1)}, \boldsymbol{\Gamma}^{(2)}$ that approximate the physical system, given with the physical hybridization functions $\Delta^R_{\text{ph}}(\omega), \Delta^K_{\text{ph}}(\omega)$ as close as possible, typically in terms of a $\mathcal{L}_2$ distance:

$$\mathcal{C}(\boldsymbol{\theta})^2 = \sum_{\alpha \in \{R,K\}} \int_{-\omega_c}^{\omega_c} \left(\mathcal{I}m \left\{ \Delta^\alpha_{\text{ph}}(\omega) - \Delta^\alpha_{\text{aux}}(\omega; \boldsymbol{\theta}) \right\} \right)^2 W(\omega)\, d\omega$$

2.3 Gradient-based Optimization

Optimization is by itself an important discipline in mathematics, thus there has been considerable effort in finding algorithms for optimizing some objective function. Without loss of generality, in the remainder of this section we will use the term optimizing interchangeably with minimizing some cost function.

In the most general case, cost functions are non-differentiable that can only be evaluated at single points in their domain. As a result, many so-called zero order optimization algorithms emerged, being for example the deterministic simplex-method or stocastic methods like variants of evolutionary strageties such as genetic algorithms, simulated annealing or paralell tempering. All of them are justified approaches to some problems. But despite their general applicability, the fact that only zero order information is utilized results in potentially intractable runtime.

Thus, many algorithms also incorporate first-order information of the cost function, the gradient, in their search schedule. This is in particular a viable solution to convex problems. When confronted with multioptima cost functions, the great disadvantage of using gradient-based methods is the problem of getting stuck in local optima as one typically follows the direction of greatest descent. In many cases, however, it is sufficient to find reasonable good local optima.

For outlining the basic procedure of gradient based methods consider the unconstrained optimization problem

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}$$

with f being the cost function that maps a parameter vector $\boldsymbol{\theta}$ to a cost value while n denotes the dimensionality of the problem. The vanilla version of a gradient-based algorithm then performs according to the following update rule:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_n}$$

with η as the update rate, usually a small number.

2.4 ADAM

The basic approach is in practice heuristically extended with a momentum that lets the parameter vector accelerate on its way down to a valley of the cost function. This idea

gave rise to more advanced gradient-based strategies such as AdaGrad [Duchi et al., 2010], RMSProp and ADAM [Kingma and Ba, 2014]. Although originally developed for stochastic gradient optimization, mentioned algorithms provide a solid foundation for solving the problem considered in this work.

Throughout this work ADAM will be used to optimize the objective function. A major advantage of this algorithm is the adaptive stepsize: ADAM was built for stochastic evaluations of the cost function. Therefore, mean and variance of each gradient component is estimated by a exponential moving average. The effective stepsize of each parameter is then inversely scaled by the variance of the gradients corresponding component. Moving averages are biased towards zero in the beginning. This is also corrected in this algorithm.

Consider for example a saddle point in the objective functions landscape. Plain stochastic gradient-based optimization would remain for a considerable time at this saddle point. But the moving average of the gradient acts like a velocity that lets the parameter vector continue to move towards better regions.

For completeness, the formal procedure of ADAM, taken from [Kingma and Ba, 2014], is given in Algorithm 1. **Remark**: ADAM is designed for stochastic optimization but the

Algorithm 1 ADAM, short for adaptive moment estimation, an algorithm for stochastic optimization. Elementwise multiplication is given with $\odot$. $\epsilon = 10^{-8}$ is to remove numerical artefacts. Default values for hyperparameters are $\alpha = 10^{-3}, \beta_1 = 0.9, \beta_2 = 0.999$

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\omega)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
  $v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t \odot g_t$ (Update biased second raw moment estimate)
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

present objective function is fully deterministic and not subject to noise influence. However, the objective function is numerically integrated and evaluated at discrete points ω which can be interpreted as a noisy gradient over many such points ω.

3 Parametrization

In the realm of computational optimization it is often crucial to find a good parametrization with respect to the degrees of freedom. For example, if the cost function is subject to some constraints, it is elegant to incorporate those restrictions into the parametrization. Mentioned approach is reminiscent of the general coordinates in the Lagrange II formalism in presence of holonomic constraints. Likewise, it is advisable to perform the parametrization in such a way that they are contributing as independent as possible to the cost function.

In the following, the chosen represantation of the auxiliary system in terms of tunable parameters shall be outlined. Both cases, symmetric and non-symmetric hybridizations are treated.

3.1 Symmetric Problems

Symmetry is determined by the physical hybridization functions. A symmetric problem is characterized by:

$$\Delta^R_{\mathrm{ph}}(-\omega) = \Delta^R_{\mathrm{ph}}(\omega)$$
$$\Delta^K_{\mathrm{ph}}(-\omega) = -\Delta^K_{\mathrm{ph}}(\omega)$$

When dealing with symmetric problems it is very useful to adapt the parametrization accordingly as the dimensionality of the optimization problem is reduced. In particular, this work will focus on optimizing the parameters of such symmetric problems. For all further considerations let n_b denote the number auxiliary bath sites. In case of symmetric problens we get $n_b \in 2\mathbb{N}$ an even number.

3.1.1 E Matrix

The parametrization of the matrix $\boldsymbol{E} \in \mathbb{R}^{(n_b+1)\times(n_b+1)}$ is straightforward by considering the tridiagonal structure with real valued entries. The $\boldsymbol{E}$ matrix in general has to fulfill following constraints:

- The matrix site corresponding to the impurity E_{ff} is identical to zero
- The secondary diagonals of $\boldsymbol{E}$ are identical:

$$E_{i,i+1} = E_{i+1,i} \qquad \forall i \in \{1, 2, ..., n_b\}$$

And in the special case of **symmetric** hybridization additional constraints need to be met:

- The main diagonal is anti-symmetric around the impurity site:

$$E_{i,i} = -E_{n_b+2-i,n_b+2-i} \qquad \forall i \in \{1, 2, ..., n_b/2\}$$

- Both of the secondary diagonals are symmetric around around the impurity site:

$$E_{i,i+1} = E_{n_b+1-i,n_b+2-i} \qquad \forall i \in \{1,2,...,n_b/2\}$$
$$E_{i+1,i} = E_{n_b+2-i,n_b+1-i} \qquad \forall i \in \{1,2,...,n_b/2\}$$

By considering the symmetry properties, we have $\boldsymbol{E}$ in the simple form of:

$$\boldsymbol{E} = \begin{pmatrix} b_1 & t_1 & 0 & \dots & & & & & \\ t_1 & b_2 & t_2 & & & & & & \\ 0 & t_2 & \ddots & \ddots & & & & & \\ \vdots & & \ddots & b_{n_b/2} & t_{n_b/2} & & & & \\ & & & t_{n_b/2} & 0 & t_{n_b/2} & & & \\ & & & & t_{n_b/2} & -b_{n_b/2} & t_{n_b/2-1} & & \vdots \\ & & & & & t_{n_b/2-1} & -b_{n_b/2-1} & \ddots & 0 \\ & & & & & & \ddots & \ddots & t_1 \\ & & & & & \dots & 0 & t_1 & -b_1 \end{pmatrix}$$

The first parameters in the $\boldsymbol{\theta}$ vector are therefore given by $\boldsymbol{\theta} = (b_1, b_2, ..., b_{n_b/2}, t_1, t_2, ..., t_{n_b/2}, ...)$

3.1.2 Γ Matrices

On the other hand the parametrization of both $\boldsymbol{\Gamma}^{(1)}$ and $\boldsymbol{\Gamma}^{(2)}$ (also with shape $(n_b+1) \times (n_b+1)$) needs to account for the hermitean and positive semi-definite properties of the respective matrix. In addition, the column and row corresponding to the impurity are set to zero for physical reasons. Moreover, in the symmetric case there is an explicit relationship between $\boldsymbol{\Gamma}^{(1)}$ and $\boldsymbol{\Gamma}^{(2)}$ which is considered by constructing only $\boldsymbol{\Gamma}^{(1)}$ and then, by the relationship, calculating $\boldsymbol{\Gamma}^{(2)}$ by means of section 3.1.5.

To let $\boldsymbol{\Gamma}$ fulfill the properties of hermiticity and positivity (semi), a way is to parametrize an auxilliary matrix H that constructs $\boldsymbol{\Gamma}$ by the mapping:

$$\boldsymbol{\Gamma} = \boldsymbol{H}\boldsymbol{H}^\dagger$$

It follows that the constraints are automatically met if additionally the row and column corresponding to the impurity are set to zero in $\boldsymbol{H}$:

positive semi-definite:

$$(\boldsymbol{a}, \boldsymbol{\Gamma}\boldsymbol{a}) = (\boldsymbol{a}, \boldsymbol{H}\boldsymbol{H}^\dagger\boldsymbol{a}) = (\boldsymbol{H}^\dagger\boldsymbol{a}, \boldsymbol{H}^\dagger\boldsymbol{a}) = (\boldsymbol{b}, \boldsymbol{b}) \geq 0, \qquad \forall \boldsymbol{a} \in \mathbb{R}^{n_b}$$

hermitean:

$$\boldsymbol{\Gamma}^\dagger = (\boldsymbol{H}\boldsymbol{H}^\dagger)^\dagger = \boldsymbol{H}\boldsymbol{H}^\dagger = \boldsymbol{\Gamma}$$

impurity row and column 0:

$$H_{ij} = 0, \qquad \text{if } i = n_0 \vee j = n_0$$
$$\Rightarrow \Gamma_{ij} = \sum_k H_{ik} H^*_{jk} = 0, \qquad \text{if } i = n_0 \vee j = n_0$$

With $(\cdot, \cdot)$ denoting the standard scalar product.

Thus, it is required to find a parametrization for H.

3.1.3 H Full Matrix Parametrization

The most intuitive way to parametrize H is to fill up the entries one by one while setting the row and column at the impurity index to zero. It is important to point out that the matrix $\mathbf{\Gamma}$ has, by its hermiticity, a total number of n_b^2 degrees of freedom (counting real and imaginary parts). Therefore, to avoid overparametrization, the matrix H is also constructed to be hermitean, resulting in the same number of independent parameters. Thus, one obtains the parametrization of H in this way to be of form:

$$\boldsymbol{H} = \begin{pmatrix} h_{1,1} & h_{1,2} & \dots & h_{1,n_b/2} & 0 & h_{1,n_b/2+1} & \dots & h_{1,n_b} \\ h_{1,2}^* & \dots & & & 0 & & & \\ \vdots & \ddots & & & \vdots & & & \\ h_{1,n_b/2}^* & \dots & & & 0 & & & \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ h_{1,n_b/2+1}^* & & & & 0 & & & \\ \vdots & & & & \vdots & & & \\ h_{1,n_b}^* & & & & 0 & & & \end{pmatrix}$$

where $h_{i,j}$ denote complex variables, except the real valued main diagonal. H exhibits a matrix shape of $(n_b + 1) \times (n_b + 1)$, the same as for the E matrix. Considering the parameter vector, one simply concatenates with these variables:

$$\boldsymbol{\theta} = (..., t_{n_b/2}, h_{1,1}, h_{2,2}, ..., \mathcal{R}e(h_{1,2}), ..., \mathcal{I}m(h_{1,2}), ...)$$

3.1.4 H Variable-Rank Parametrization

In order to infer the importance of the number of parameters that are employed by the complete specification of the $\mathbf{\Gamma}$ matrix, a new parametrization method is utilized that allows to specify the possible maximum rank of $\mathbf{\Gamma}$, and thus the number of parameters, of the $\boldsymbol{H}$ matrix. The new formulation of the parametrization of $\boldsymbol{H}$ is reminiscent of the eigenvector decomposition of matrices with the eigenvalues being absorbed into the eigenvectors:

$$\mathbf{\Gamma} = \sum_{i=1}^{M} h_i \cdot h_i^\dagger = \boldsymbol{H}\boldsymbol{H}^\dagger$$

with M being the maximum allowed rank and $h_i = (h_{1,i}, h_{2,i}, \dots, 0, \dots, h_{2n_b,i})^T$ denoting the complex vector with index i. Thus, we have for the elements of the matrix $\boldsymbol{H}$:

$$H_{ij} = h_{i,j}$$

By given formulation, this compares well to the Full Matrix Parametrization in 3.1.3 with H having a shape of $(n_b + 1) \times M$. This similarity will come in handy for the gradient formulation.

In practice, the implemented parameter vector data structure could order the components in given way:

$$\boldsymbol{\theta} = (..., b_{n_b/2}, \mathcal{R}e(h_{1,1}), ..., \mathcal{R}e(h_{M,n_b}), \mathcal{I}m(h_{1,1}), ..., \mathcal{I}m(h_{M,n_b}))$$

By setting $M = n_b/2$, one obtains the same expressiveness as in the full matrix parametrization.

In numerical mathematics this kind of approximation is also termed as **hierarchical matrix** or shortly, and also coinciding with the notation in this work, as H-matrix. An in-depth consideration of such matrices is given in [Bebendorf, 2008].

3.1.5 Relationship of the two Γ matrices

In case of symmetry, the $\boldsymbol{\Gamma^{(2)}}$ matrix is related to $\boldsymbol{\Gamma^{(1)}}$ by:

$$\Gamma^{(2)}_{ij} = (-1)^{i+j}\ \Gamma^{(1)}_{n-j+1,n-i+1}$$

with $n = n_b + 1$ the size of the matrix $\Gamma^{(1)}$ in any direction.

3.2 General, Non-symmetric Problems

In non-symmetric, general hybridizations there are two differences in the parametrization:

- $\boldsymbol{\Gamma^{(1)}}, \boldsymbol{\Gamma^{(2)}}$ are to be parametrized independent of each other
- $\boldsymbol{E}$ has not the symmetry around the impurity, resulting in a parametrization of:

$$\boldsymbol{E} = \begin{pmatrix}
b_1 & t_1 & 0 & \dots & & & & & \\
t_1 & b_2 & t_2 & & & & & & \\
0 & t_2 & \ddots & \ddots & & & & & \\
\vdots & & \ddots & b_{n_b/2} & t_{n_b/2} & & & & \\
 & & & t_{n_b/2} & 0 & t_{n_b/2} & & & \\
 & & & & t_{n_b/2+1} & b_{n_b/2+1} & t_{n_b/2+2} & & \vdots \\
 & & & & & t_{n_b/2+2} & b_{n_b/2+2} & \ddots & 0 \\
 & & & & & & \ddots & \ddots & t_{n_b} \\
 & & & & & \dots & 0 & t_{n_b} & b_{n_b}
\end{pmatrix}$$

4 Derivation of the Gradient

In order to perform a gradient-based optimization on the cost function given in Section 2.2, it is required to find an expression for the gradient with respect to the auxiliary system parameters: $\boldsymbol{\nabla_\theta}\ \mathcal{C}(\boldsymbol{\theta})^2$.

Since the cost function depends on $\boldsymbol{\theta}$ through various steps of computation, it is advisable to make appropriate use of the chain rule to compute the gradient all the way back. Although we need to deal with complex functions, we do not require them to be holomorphic. Instead, they are treated as multivariate mappings $\mathbb{R}^n \to \mathbb{R}$. This way, occuring functional relationships are not required to fulfill the Cauchy-Riemann equations, e.g. complex absolute value whose derivative with respect to its arguments is required later. We start with the cost function itself:

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\mathcal{C}^2 = \frac{\partial}{\partial\theta_i}\sum_{\alpha\in\{R,K\}}\int_{-\omega_c}^{\omega_c}\left(\mathcal{I}m\left\{\Delta^\alpha_{\text{ph}}(\omega) - \Delta^\alpha_{\text{aux}}(\omega;\boldsymbol{\theta})\right\}\right)^2 d\omega \overset{\text{Leibnitz}}{=} \\ \sum_{\alpha\in\{R,K\}}\int_{-\omega_c}^{\omega_c}\frac{\partial}{\partial\theta_i}\left(\mathcal{I}m\left\{\Delta^\alpha_{\text{ph}}(\omega) - \Delta^\alpha_{\text{aux}}(\omega;\boldsymbol{\theta})\right\}\right)^2 d\omega = \\ -2\sum_{\alpha\in\{R,K\}}\int_{-\omega_c}^{\omega_c}\mathcal{I}m\left\{\Delta^\alpha_{\text{ph}}(\omega) - \Delta^\alpha_{\text{aux}}(\omega;\boldsymbol{\theta})\right\}\mathcal{I}m\left(\frac{\partial}{\partial\theta_i}\Delta^\alpha_{\text{aux}}(\omega,\boldsymbol{\theta})\right) d\omega\end{aligned}$$

The next dependency on the parameter vector is thus given by Δ^R_{aux} and Δ^K_{aux} of which only the imaginary parts will be required. We proceed with the derivative of the hybridization functions:

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\Delta^R_{\text{aux}} &= \frac{\partial}{\partial\theta_i}\left(\frac{1}{g^R(\omega)} - \frac{1}{G^R_{ff}(\omega;\boldsymbol{\theta})}\right) = \left(G^R_{ff}(\omega;\boldsymbol{\theta})\right)^{-2}\frac{\partial}{\partial\theta_i}G^R_{ff}(\omega;\boldsymbol{\theta}) \\ \frac{\partial}{\partial\theta_i}\Delta^K_{\text{aux}} &= \frac{\partial}{\partial\theta_i}\left(\frac{G^K_{ff}(\omega;\boldsymbol{\theta})}{|G^R_{ff}(\omega;\boldsymbol{\theta})|^2}\right) \\ &= \frac{1}{|G^R_{ff}(\omega;\boldsymbol{\theta})|^2}\frac{\partial G^K_{ff}(\omega;\boldsymbol{\theta})}{\partial\theta_i} - 2\frac{G^K_{ff}(\omega;\boldsymbol{\theta})}{|G^R_{ff}(\omega;\boldsymbol{\theta})|^4}\mathcal{R}e\left\{\left(G^R_{ff}(\omega;\boldsymbol{\theta})\right)^*\frac{\partial}{\partial\theta_i}G^R_{ff}(\omega;\boldsymbol{\theta})\right\}\end{aligned}$$

The occuring complex absolut value zz^* is not complex differentiable. However, because we see it as a complex valued multivariate function, a derivative with respect to these arguments is valid. To see this, let $f : \mathbb{R}^n \to \mathbb{C}$ be such a mapping:

$$\frac{\partial}{\partial x_i}\left(f(\boldsymbol{x})f^*(\boldsymbol{x})\right) = \left(\frac{\partial}{\partial x_i}f(\boldsymbol{x})\right)f^*(\boldsymbol{x}) + f(\boldsymbol{x})\left(\frac{\partial}{\partial x_i}f^*(\boldsymbol{x})\right) = \mathcal{R}e\left\{f^*(\boldsymbol{x})\frac{\partial}{\partial x_i}f(\boldsymbol{x})\right\}$$

Since G^α_{ff} is an element of a matrix, it is required to find the derivative of corresponding matrices.

4.1 Derivative of G^R_{ff}

We resume:

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\boldsymbol{G}^R(\omega;\boldsymbol{\theta}) &= \frac{\partial}{\partial\theta_i}\left(\omega\mathbb{1} - \boldsymbol{E}(\boldsymbol{\theta}) + i\left(\boldsymbol{\Gamma}^{(1)}(\boldsymbol{\theta}) + \boldsymbol{\Gamma}^{(2)}(\boldsymbol{\theta})\right)\right)^{-1} \\ &=: \frac{\partial}{\partial\theta_i}\left(\boldsymbol{M}(\omega;\boldsymbol{\theta})\right)^{-1} = -\boldsymbol{M}(\omega;\boldsymbol{\theta})^{-1}\left(\frac{\partial}{\partial\theta_i}\boldsymbol{M}(\omega;\boldsymbol{\theta})\right)\boldsymbol{M}(\omega;\boldsymbol{\theta})^{-1} \\ &= -\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\left(\frac{\partial}{\partial\theta_i}\boldsymbol{M}(\omega;\boldsymbol{\theta})\right)\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\end{aligned}$$

since it holds:

$$0 = \left(\boldsymbol{A}\boldsymbol{A}^{-1}\right)' = \boldsymbol{A}'\boldsymbol{A}^{-1} + \boldsymbol{A}\left(\boldsymbol{A}^{-1}\right)' \Rightarrow \left(\boldsymbol{A}^{-1}\right)' = -\boldsymbol{A}^{-1}\boldsymbol{A}'\boldsymbol{A}^{-1}$$

Next step is to identify the correct derivatives of the just defined matrix $\boldsymbol{M}$.

$$\frac{\partial}{\partial\theta_i}\boldsymbol{M}(\boldsymbol{\theta}) = \begin{cases} -\frac{\partial}{\partial\theta_i}\boldsymbol{E}(\boldsymbol{\theta}) & \text{if } \theta_i \text{ parametrizes } \boldsymbol{E} \\ i\frac{\partial}{\partial\theta_i}\boldsymbol{\Gamma}^{(\alpha)}(\boldsymbol{\theta}) & \text{if } \theta_i \text{ parametrizes } \boldsymbol{\Gamma}^{(\alpha)} \end{cases}$$

4.2 Derivative of G^K_{ff}

$$\begin{aligned}
\frac{\partial}{\partial\theta_i}\boldsymbol{G}^K(\omega;\boldsymbol{\theta}) &= \frac{\partial}{\partial\theta_i}\left(2i\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\left(\boldsymbol{\Gamma}^{(2)}(\boldsymbol{\theta})-\boldsymbol{\Gamma}^{(1)}(\boldsymbol{\theta})\right)\boldsymbol{G}_A(\omega;\boldsymbol{\theta})\right) \\
&= \frac{\partial}{\partial\theta_i}\left(2i\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\left(\boldsymbol{\Gamma}^{(2)}(\boldsymbol{\theta})-\boldsymbol{\Gamma}^{(1)}(\boldsymbol{\theta})\right)[\boldsymbol{G}_R(\omega;\boldsymbol{\theta})]^\dagger\right) \\
&= 2i\left(\frac{\partial}{\partial\theta_i}\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\right)\left(\boldsymbol{\Gamma}^{(2)}-\boldsymbol{\Gamma}^{(1)}\right)[\boldsymbol{G}^R(\omega;\boldsymbol{\theta})]^\dagger \\
&\quad + 2i\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\left(\boldsymbol{\Gamma}^{(2)}-\boldsymbol{\Gamma}^{(1)}\right)\left[\frac{\partial}{\partial\theta_i}\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\right]^\dagger \\
&\quad + 2i\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\left(\frac{\partial}{\partial\theta_i}\boldsymbol{\Gamma}^{(2)}(\boldsymbol{\theta})-\frac{\partial}{\partial\theta_i}\boldsymbol{\Gamma}^{(1)}(\boldsymbol{\theta})\right)[\boldsymbol{G}^R(\omega;\boldsymbol{\theta})]^\dagger
\end{aligned}$$

$$= 2i\begin{cases}
\left(\frac{\partial}{\partial\theta_i}\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\right)\left(\boldsymbol{\Gamma}^{(2)}-\boldsymbol{\Gamma}^{(1)}\right)[\boldsymbol{G}^R(\omega;\boldsymbol{\theta})]^\dagger & \\
+\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\left(\boldsymbol{\Gamma}^{(2)}-\boldsymbol{\Gamma}^{(1)}\right)\left[\frac{\partial}{\partial\theta_i}\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\right]^\dagger & \text{if } \theta_i \text{ parametrizes } \boldsymbol{G}^R(\omega;\boldsymbol{\theta}) \\
-\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\left(\frac{\partial}{\partial\theta_i}\boldsymbol{\Gamma}^{(1)}(\boldsymbol{\theta})\right)[\boldsymbol{G}^R(\omega;\boldsymbol{\theta})]^\dagger & \text{if } \theta_i \text{ parametrizes } \boldsymbol{\Gamma}^{(1)}(\boldsymbol{\theta}) \\
\boldsymbol{G}^R(\omega;\boldsymbol{\theta})\left(\frac{\partial}{\partial\theta_i}\boldsymbol{\Gamma}^{(2)}(\boldsymbol{\theta})\right)[\boldsymbol{G}^R(\omega;\boldsymbol{\theta})]^\dagger & \text{if } \theta_i \text{ parametrizes } \boldsymbol{\Gamma}^{(2)}(\boldsymbol{\theta})
\end{cases}$$

4.3 Derivative of E

The derivative of $\boldsymbol{E}$ is elementwise and thus it is typically a matrix with a few non-zero entries being ± 1. Likewise, in the symmetric case, the same parameter occurs always at least two times. Suppose for example θ_k is the parameter that controls the transition probability of the second bath site. Then we would obtain:

$$\frac{\partial}{\partial\theta_k}\boldsymbol{E}(\boldsymbol{\theta}) = \begin{pmatrix} 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & \dots \\ 0 & 1 & 0 & \dots & \\ 0 & 0 & 0 & & \\ \vdots & \vdots & & \ddots & \end{pmatrix}$$

4.4 Derivative of Γ^α

Because $\mathbf{\Gamma}$ was defined by a mapping $\mathbf{\Gamma} = \boldsymbol{H}\boldsymbol{H}^\dagger$ (in both the Full Matrix parametrization given in Section 3.1.3 as well as in the case of variable-rank parametrization given in Section 3.1.4), we need to further differentiate in order to get to the parameters.

$$\frac{\partial}{\partial\theta_i}\mathbf{\Gamma}^\alpha(\boldsymbol{\theta}) = \frac{\partial}{\partial\theta_i}\left(\boldsymbol{H}(\boldsymbol{\theta})\boldsymbol{H}(\boldsymbol{\theta})^\dagger\right) = \left(\frac{\partial}{\partial\theta_i}\boldsymbol{H}(\boldsymbol{\theta})\right)\boldsymbol{H}(\boldsymbol{\theta})^\dagger + \boldsymbol{H}(\boldsymbol{\theta})\left(\frac{\partial}{\partial\theta_i}\boldsymbol{H}(\boldsymbol{\theta})\right)^\dagger$$

And $\frac{\partial}{\partial\theta_i}\boldsymbol{H}$ is again a very sparse matrix with at most 2 non-zero entries. Note that also complex values occur.

4.5 Full Gradient

Since every derivative is known by now, one can compute the gradient. The procedure works step by step. Computation starts at the first level of parameter occurences. The sequence of computation is then given by:

$$\frac{\partial}{\partial\theta_i}\boldsymbol{H}, \frac{\partial}{\partial\theta_i}\boldsymbol{E} \to \frac{\partial}{\partial\theta_i}\mathbf{\Gamma}^\alpha \to \frac{\partial}{\partial\theta_i}\boldsymbol{G}^R \to \frac{\partial}{\partial\theta_i}\boldsymbol{G}^K \to \frac{\partial}{\partial\theta_i}\Delta^R_{\text{aux}}, \frac{\partial}{\partial\theta_i}\Delta^K_{\text{aux}} \to \frac{\partial}{\partial\theta_i}\mathcal{C}^2 \quad \forall i$$
$$\to \nabla_{\boldsymbol{\theta}}\, \mathcal{C}^2$$

5 Implementation

The expressions that construct the gradient of the cost function are then required to be implemented on a digital computer with finite precision in order to use it for numerical optimization.

However, it is not only required to type in the equations for the gradient but one also needs to take care of several other problems that arise by conversion to finite precision. In addition, one has to keep in mind the duration of computation, since the gradient-enhanced optimization was introduced for speedup in the first place.

To instruct a computer with operations, a programming language is required. In this work we rely on a `Python` implementation to be flexible. Since matrix computations are involved the packages `numpy` and `scipy` are appropriate extensions to the standard functionality.

5.1 Numerical Integration

Problems start with an integral over a range $\omega \in (-\omega_c, \omega_c)$ occuring in the gradient expression. By nature of numerics, this needs to be treated in an according way. Ultimately, the integral will be converted to a finite sum with appropriate step sizes. Some of the simplest methods to treat integrals numerically are given with quadrature formulas:

- Constant approximation (Rectangle rule)
- Linear approximation (Trapez rule)
- Quadratic approximation (Simpson rule)

In this problem, the constant approximation is utilized justified by the argument that not the exact value of the gradient is required but instead, mostly the **right direction**. Therefore, the interval $(-\omega_c, \omega_c)$ is split into equal pieces with one sample point at which evaluation takes place.

$$\int_{-\omega_c}^{\omega_c} f(\omega)\, d\omega \to \sum_{i=1}^{N} f(2\omega_c \frac{i}{N} - \omega_c)\ \frac{2\omega_c}{N}$$

As a result, the derivatives of $\boldsymbol{G}^R, \boldsymbol{G}^K$ have to be carried out over multiple ω for each gradient computiation of the cost function.

By intuition, the precision of the gradient increases as N grows. As a rule of thumb, N has to be so large that $\Delta\omega = \frac{2\omega_c}{N}$ is able to capture salient features of the hybridization function.

5.2 Matrix Inversion

Consider $\boldsymbol{G}^R(\omega; \boldsymbol{\theta})$. The vanilla version of gradient computation would require that at each different ω, a matrix inversion is performed. Inverting matrices is not only a very time consuming operation but it can also be numerically instable depending on the condition of respective matrix.

A simple consideration that exploits the structure of the problem contributes to speed up computation. We have:

$$\begin{aligned}
\boldsymbol{G}^R &= (\omega\mathbb{1} + \boldsymbol{X})^{-1} && \text{with } \boldsymbol{X} := -\boldsymbol{E} + i(\boldsymbol{\Gamma}^{(1)} + \boldsymbol{\Gamma}^{(2)}) \\
\boldsymbol{X} &= \boldsymbol{V}^\dagger \boldsymbol{D} \boldsymbol{V} && (\text{Diagonalization},\ \boldsymbol{X} = \boldsymbol{X}^\dagger) \\
\boldsymbol{G}^R &= \left(\omega\mathbb{1} + \boldsymbol{V}^\dagger \boldsymbol{D} \boldsymbol{V}\right)^{-1} \\
&= \left(\omega\mathbb{1}\boldsymbol{V}^\dagger \boldsymbol{V} + \boldsymbol{V}^\dagger \boldsymbol{D} \boldsymbol{V}\right)^{-1} \\
&= \left(\boldsymbol{V}^\dagger \omega\mathbb{1} \boldsymbol{V} + \boldsymbol{V}^\dagger \boldsymbol{D} \boldsymbol{V}\right)^{-1} && (\boldsymbol{V} \text{ commutes with } \mathbb{1}) \\
&= \boldsymbol{V}^\dagger \left(\omega\mathbb{1} + \boldsymbol{D}\right)^{-1} \boldsymbol{V}
\end{aligned}$$

As a result once the eigenvalue decomposition is computed for $\boldsymbol{X}$, we are left with a much simpler problem of inverting a diagonal matrix which is carried out elementwise and thus much faster. Full matrix inversion is usually $\mathcal{O}(n^3)$ (cubic in matrix length) depending on the algorithm while inversion of a diagonal matrix is $\mathcal{O}(n)$ (linear in matrix length) plus the additional eigenvalue decomposition taking place **once** in the numerical integration.

5.3 Sanity Check with Finite Differences

After implementing the mentioned computations, it is good practice to check for sanity. Although one cannot numerically prove correctness, one can clearly determine if the computation has a flaw. The procedure conducted is simple and works with just a few steps according to Algorithm 2.

Algorithm 2 Sanity check of gradient

Input $\boldsymbol{x}, f$
$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}}\, \mathcal{C}(\boldsymbol{x})^2$
$\tilde{\boldsymbol{g}} \leftarrow 0$
for $\theta_i \in \boldsymbol{\theta}$ **do**
$\tilde{g}_i \leftarrow \Big(f(x_i + \epsilon) - f(x_i - \epsilon) \Big) / \, (2\epsilon)$
end for
return $\begin{cases} \text{true} & \tilde{\boldsymbol{g}} \approx \boldsymbol{g} \\ \text{false} & \text{else} \end{cases}$

The given check indeed suggested that the gradient implementation is correct.

5.4 Tensorflow

Although implemented, a plain Python implementation with numpy, scipy etc. was too slow for practical improvements (single core). Thus, for optimizing the parameters in a fast fashion, we employ the **Tensorflow** package of Google [Abadi et al., 2016]. Its purpose is originally to aid construction of large-scale machine learning systems by computational graphs.

Such a computation graph represents a computation by means of many nodes connected by edges. Each node itself represents a mathematical operation and processes its inputs, typically a multidimensional array, to give outputs, also multidimensional arrays. In particular, the graph itself has to be acyclic and is processed feedforward. The edges in the graph are representative for computational dependencies. Given a mathematical expression that is composed of many operations, in our case for example the cost function, we may create a computational graph representing the equations.

Each node represents an operation and thus, the partial derivatives of the output of the node with respect to its inputs can be specified symbolically, if existent. The advantage of such computational graphs resides in the fact that the derivative of a node can also be included as a node in the graph, depending on the same inputs of the differentiated node. Furthermore, the chain rule of calculus may subsequently be applied automatically to give the derivative of the entire computation graph. Thus, **by specifying the computation graph of a composed mathematical operation, the gradient of this operation is simply an extension to the graph**.

Tensorflow and variants are typically used in machine learning applications. A typical problem in this area is classification. Given a data set consisting of examples $\boldsymbol{x}_i \in X$ and labels $t_i \in T$ one needs to find a mapping $f : X \to T$ that maps the examples

accurately to the target labels, generalizing to unseen similar samples. This mapping f can be represented as an artificial neural network with a huge number of free parameters. And again can be represented as a computation graph that allows to find the gradient of some cost function with respect to the free parameters. Since the problem that we are confronted with is similar to that, a computation graph approach is suitable.

To provide a simple example of a computation graph consider:

$$f(a) := c(a)d = abd$$
$$\frac{\partial}{\partial a} f(a) = \frac{\partial f}{\partial c}\frac{\partial c}{\partial a} = d\frac{\partial c}{\partial a} = db$$

This simple example demonstrates the chain rule which is automatically computed by tensorflow.To illustrate the corresponding computation graph consider Figure 1. A simplified algorithm that computes the derivatives in a computational graph can be taken from [Goodfellow et al., 2016], Algorithm 6.5 and Algorithm 6.6.

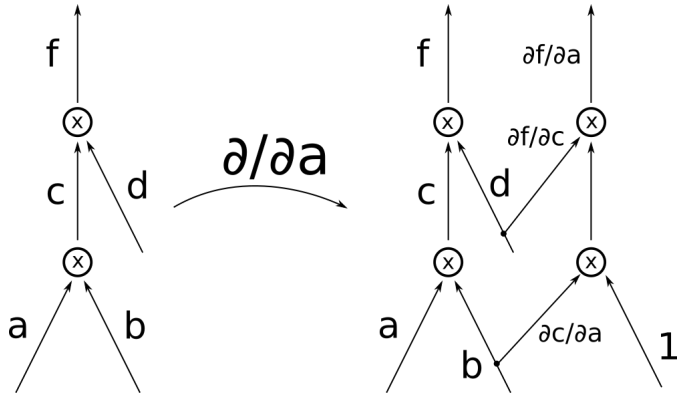

Figure 1: A simple example for a computation graph that is extended by a derivation with respect to some parameters

However, complex numbers are not yet well supported in symbolic gradient computation. It is therefore required to pass all imaginary and real parts along seperately while formulating the complex field multiplication and other complex operations manually in order to keep the gradient derivative ability of tensorflow. Consider for example inverting a complex matrix. In tensorflow, to be able to correctly propagate the derivative, it is required to build the inverse by solving a linear system of equations. Extending this operation to the complex domain can be achieved by solving a bigger system. Suppose $\boldsymbol{M} = (\boldsymbol{A} + i\boldsymbol{B}) \in \mathcal{C}^{n\times n}$ is the complex matrix we wish to invert. Further assume that $\boldsymbol{y}_k$ is the k-th column vector in the identity matrix, i.e.: $(\boldsymbol{y}_1|\boldsymbol{y}_2|...|\boldsymbol{y}_n) = \mathbb{1}$. Inversion can be done by solving:

$$\boldsymbol{M}\boldsymbol{x}_k = \boldsymbol{y}_k \qquad\qquad k \in \{1, 2, ..., n\}$$

Additionally it is required to map given complex equations to the real domain. Consider:

$$(\boldsymbol{A} + i\boldsymbol{B})\boldsymbol{x}_k = \boldsymbol{y}_k + i\boldsymbol{0} \qquad\qquad \boldsymbol{x}_k = \boldsymbol{a}_k + i\boldsymbol{b}_k$$
$$(\boldsymbol{A} + i\boldsymbol{B})(\boldsymbol{a}_k + i\boldsymbol{b}_k) = \boldsymbol{y}_k + i\boldsymbol{0}$$
$$\boldsymbol{A}\boldsymbol{a} - \boldsymbol{B}\boldsymbol{b} + i(\boldsymbol{B}\boldsymbol{a} + \boldsymbol{A}\boldsymbol{b}) = \mathbb{1} + i\boldsymbol{0}$$

Rewriting this as matrix equation, while considering that both real and imaginary parts have to independently fulfill the equations, leads to the real valued equation system of:

$$\begin{pmatrix} \boldsymbol{A} & -\boldsymbol{B} \\ \boldsymbol{B} & \boldsymbol{A} \end{pmatrix} \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{pmatrix} = \begin{pmatrix} \boldsymbol{y}_k \\ \boldsymbol{0} \end{pmatrix}$$

6 Results

After implementation, the gradient-enhanced optimization with the ADAM algorithm, described in Section 2.4 as well as in [Kingma and Ba, 2014], was subject to testing. Both the full matrix and the variable-rank parametrization of the $\boldsymbol{H}$ matrix was evaluated.

The system on which all experiments were conducted is driven by an Intel i7-3630QM quadcore CPU.

6.1 Reference Physical System

To perform a fit, a reference system to be approximated is required. The testing hybridization was given by the following retarded and Keldysh components:

$$\begin{aligned} \Phi(\omega;\mu,T) &:= \left(\exp\left(\frac{\omega-\mu}{T}\right)+1\right)^{-1} \\ \Delta^R_{ph}(\omega) &:= -i\pi\left(\frac{2D}{\pi}\right)^2 \frac{1}{2D}\Phi(\omega;D,T)\Phi(\omega;D,T) \\ \Delta^K_{ph}(\omega) &:= 2\Delta^R_{ph}(\omega)\left(1-\Phi(\omega;\mu_L,T)-\Phi(\omega;\mu_R,T)\right) \\ &\text{with} \\ \mu_L &= -3 \\ \mu_R &= 3 \\ D &= 10 \\ T &= 0.3 \end{aligned}$$

This test hybridization function, in the following referred to as physical system, is illustrated in Figure 2. As mentioned earlier, this work focuses on symmetric problems like the given test hybridization.

6.2 Reference Solution - Parallel Tempering

The performance of this optimization method is compared to an already implemented optimization algorithm, parallel tempering, in terms of speed and misfit.

Parallel Tempering is an optimization algorithm that is based upon Simulated Annealing. The extension is that many copies of Simulated Annealing with different temperatures run and exchange their configurations accordingly. The advantage of which

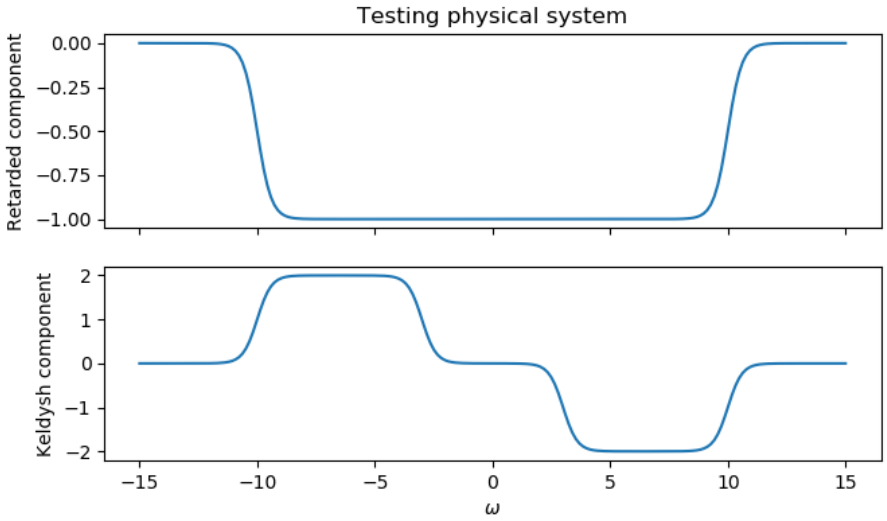

Figure 2: The testing physical system. The upper plot is the retarded component while the lower one illustrates the Keldysh part

is that providing enough runtime, the algorithm will converge to the global optimum. However, it is not the main target of this work to describe the working blocks of this stochastic optimization algorithm but to compare against it. In particular various numbers of auxiliary bath sites are tried: 2, 4, 6. With more and more bath sites, a drastic increase in runtime is observed, *curse of dimensionality*, see Table 1.

The reference solutions that parallel tempering found are illustrated in Figure 3. Correspondingly, actual cost function values and runtime is given in Table 1. As one can easily determine, the physical system is only mapped properly to an auxiliary system with at least 6 bath sites. On the other hand, the execution time of parallel tempering with this auxiliary system size took approximately 2.75 hours.

Table 1: Cost function values of the final solution of parallel tempering with various numbers of bath sites. In addition, execution runtime is given.

$\mathcal{C}^2_{\text{P.t.}}$ ... Cost function value of parallel tempering (comparison)
T ... Execution time

	$\mathcal{C}^2_{\text{P.t.}}$	T/s
2 Bath Sites	$192.1 \cdot 10^{-3}$	125
4 Bath Sites	$53.18 \cdot 10^{-3}$	453
6 Bath Sites	$0.683 \cdot 10^{-3}$	9949

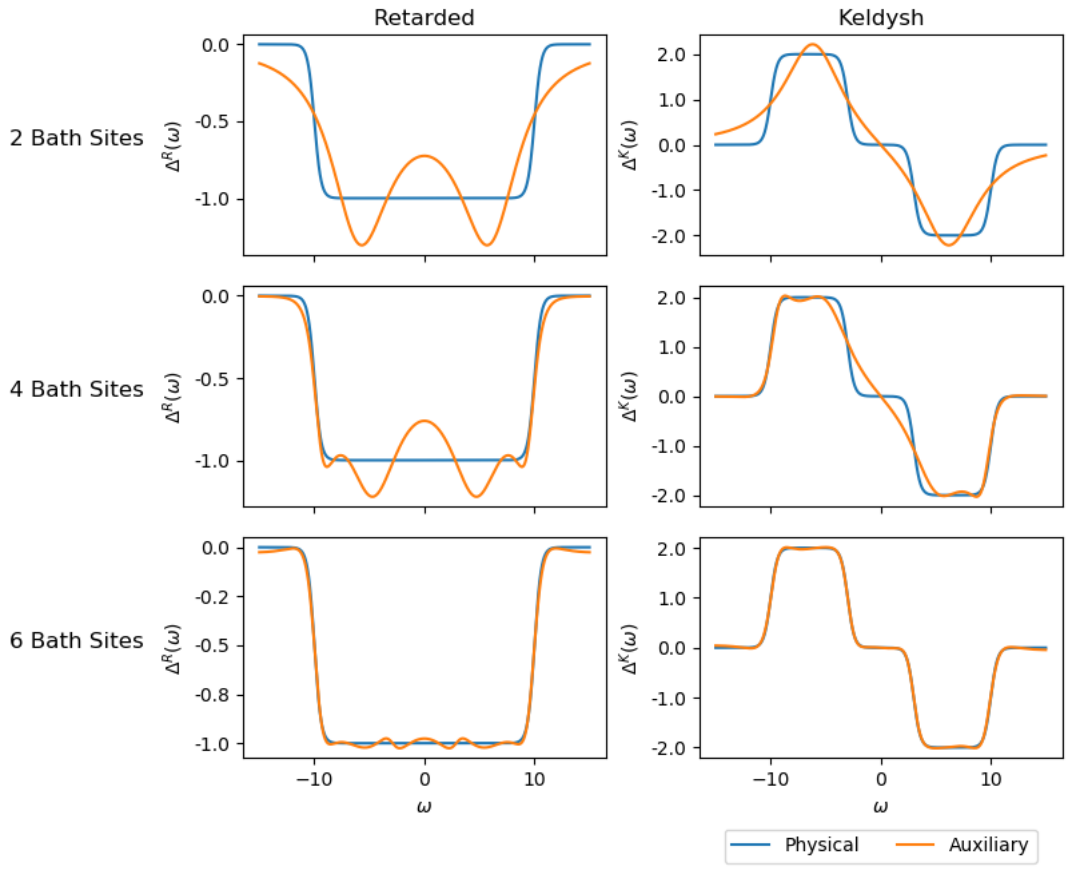

Figure 3: The solutions found by parallel tempering compared to the original physical system. Each row represents a different size of the auxiliary system. Left column: Retarded component of hybridization, Right column: Keldysh part.

6.3 Reference Solution - Nonimprovable

To see wether the reference solution is a local optimum and to support the gradient-based optimizations correctness, it is necessary to show that the optimizer implemented in this work does not move the parameters away from the found solution if initialized with it. The implementation of parallel tempering for AMEA is based on the full matrix parametrization described in Section 3.1.3. Thus, only this way of parametrization can be subject to this analysis.

Indeed it was found that after initializing the gradient-based method with the final solution obtained by parallel tempering, the algorithm could not further improve this solution. Figure 4 suggests that only up to little deviations, likely to be introduced by the step size of parallel tempering, gradient-based algorithms cannot exhibit further significant improvements to the solutions of parallel tempering. Since the existing optimizer is trusted to be correct, the result indicates that also the gradient-based optimizer implemented in this work is correct in terms of finding local optima.

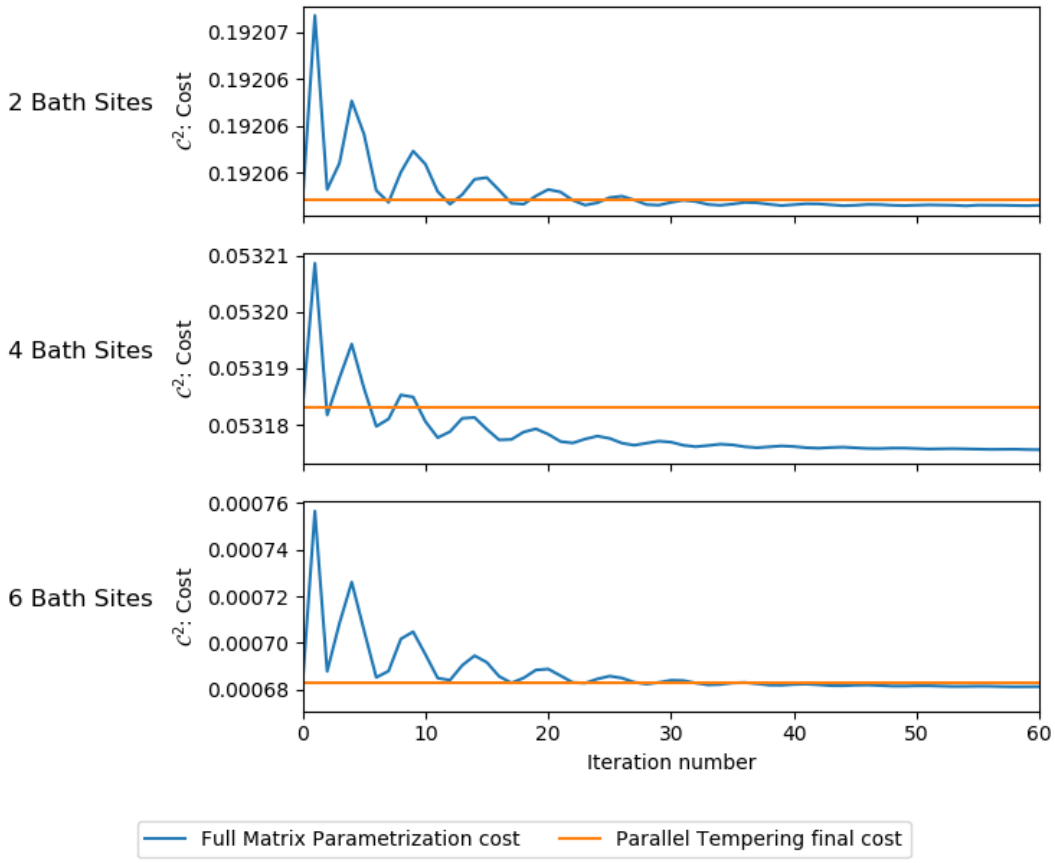

Figure 4: The cost functions evolution after the gradient-based optimizer is initialized with the parallel tempering solutions. Each row represents a different size of the auxiliary system. Oscillations are introduced also because of the gradient normalization dynamics of the ADAM algorithm.

6.4 Optimizing with Full Matrix Parametrization

This section describes the optimizers performance in the case of the full matrix parametrization given in Section 3.1.3. The parameter vector has the same structure as in the implementation of parallel tempering that we wish to compare to. During the tests it turned out that the problems structure is not convex and there are many local minima in which our gradient-based optimizer can get stuck. At this point, it shall already be emphasized that this algorithm should **not perform on its own** to provide reliable good results.

The testing procedure involves random initialization of the parameters (gaussian, zero mean and 0.1 standard deviation) and then applying gradient-descent on the cost function with mentioned ADAM algorithm. Since random initializations are being used, 5 identical runs with different initializations are executed.

The gradient-based optimizer using this parametrization is only able to find the solution found by parallel tempering in the case of 2 bath sites. In every setting that uses

a more complex representation of the auxiliary system, 4, 6 or more auxiliary bath sites, the gradient-based optimizer does not reliably find an appropriate solution that captures the main features of the physical system we try to approximate. However, in most of the cases the solution is still acceptable, at least for the considered testing physical hybridization function. But, for example in case of 6 bath sites, the random initalization was attracted to a local optimum, consider Figure 6.

In Table 2 we summarize the final cost function value and execution runtime of the gradient-based optimizer depending on the number of bath sites used in the auxiliary system representation. Gradient-based optimization was considered to be done if the cost function did not improve 1% for at least 500 consecutive parameter update iterations. For comparison see Table 1 that enumerates the parallel tempering solutions. In addition, Figure 5 demonstrates various realizations of the cost functions evolution depending on the iteration number.

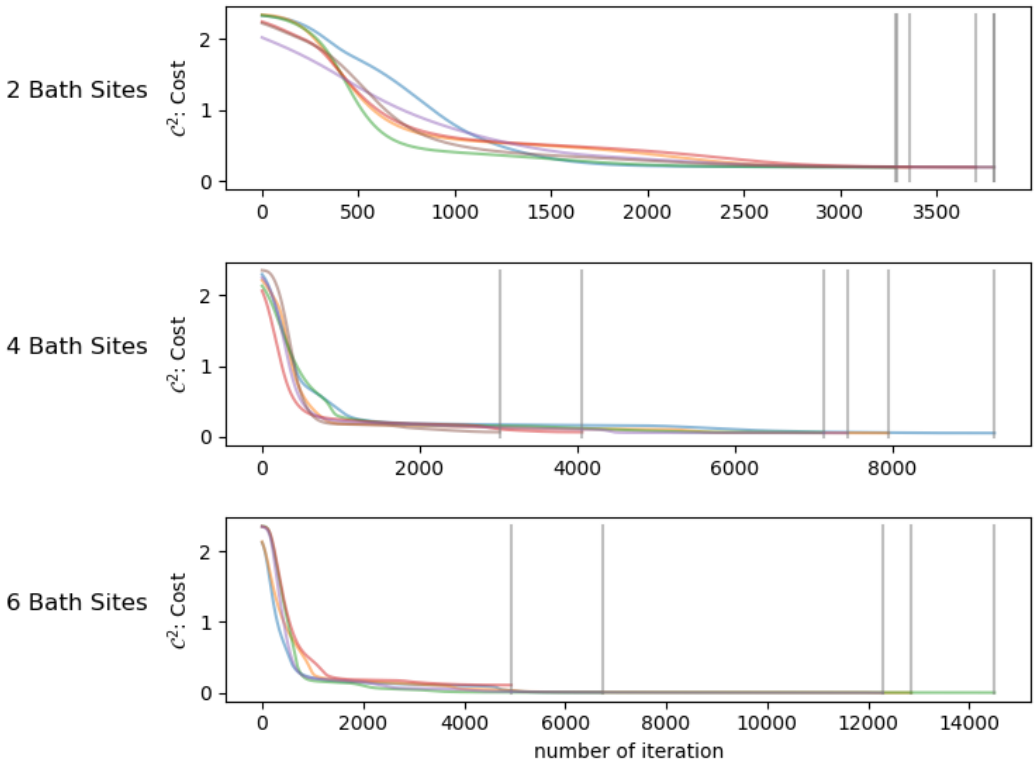

Figure 5: Optimization trial realizations with different auxiliary system sizes. Each graph corresponds to a seperate optimization run. Grey bars indicate the stopping of a run if no further improvement (1% over 500 consecutive iterations) was achieved.

As a main consequence of the many local optima, this mode of operation should not be used on itself to provide reliable results. Instead, it is suggested that the existing parallel tempering algorithm is enhanced in the lowest temperature instance by exchanging the stochastic parameter update with the gradient-based method.

Table 2: Cost function values of the final solution of gradient-based optimization with full matrix parametrization. In addition, execution runtime is given.

$\min(\mathcal{C}^2_{\text{grad,f}})$ ... Minimal cost function value of the gradient-based optimizer
$\text{avg}(\mathcal{C}^2_{\text{grad,f}})$ ... Average of the cost function value of the gradient-based optimizer
$\mathcal{C}^2_{\text{P.t.}}$ ... Cost function value of parallel tempering (comparison)
T ... Execution time

	$\min(\mathcal{C}^2_{\text{grad,f}})$	$\text{avg}(\mathcal{C}^2_{\text{grad,f}})$	$\mathcal{C}^2_{\text{P.t.}}$	T/s
2 Bath Sites	$192.1 \cdot 10^{-3}$	$192.7 \cdot 10^{-3}$	$192.1 \cdot 10^{-3}$	57 ± 4
4 Bath Sites	$53.18 \cdot 10^{-3}$	$59.19 \cdot 10^{-3}$	$53.18 \cdot 10^{-3}$	185 ± 63
6 Bath Sites	$0.676 \cdot 10^{-3}$	$23.87 \cdot 10^{-3}$	$0.683 \cdot 10^{-3}$	502 ± 183

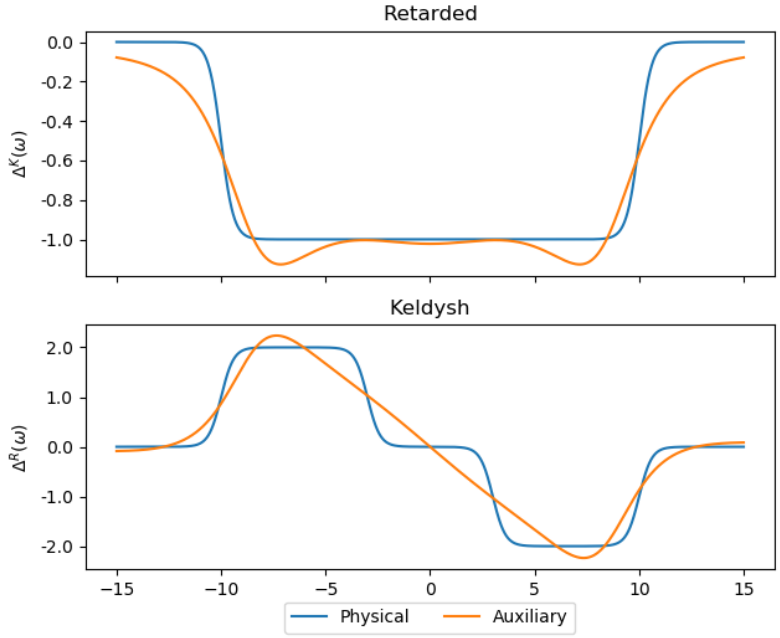

Figure 6: Optimization arrived at a local optimum with full matrix parametrization. This example is with 6 auxiliary bath sites and should be able to exhibit the salient features of the physical hybridization function.

6.5 Optimizing with Variable Rank Parametrization

To use gradient-based methods as a standalone optimization technique, a different solution is required. Indeed, during this work another approach led to success. Based on the variable-rank parametrization using H-matrices given in Section 3.1.4, it is possible to iteratively enhance model complexity while using previous parameter settings as initializations.

Recall that the parametrization in this case is given by

$$\boldsymbol{\Gamma} = \sum_{i=1}^{M} h_i \cdot h_i^{\dagger} = \boldsymbol{H}\boldsymbol{H}^{\dagger},$$

with M denoting the maximum rank of the $\boldsymbol{H}$ matrix. Then, suppose an optimization is performed with $M = M_{\text{start}}$. After convergence, M is increased by 1, keeping the previous h_i as initializations. A detailed description is given in Algorithm 3.

Algorithm 3 Iterative optimization

```
M ← M_start
H ← rand() ∈ ℂ^{N×M_start}
for M ∈ {M_start + 1, M_start + 2, ..., M_end} do
    h_M ← 0
    H ← (H|h_M)
    Optimize Model
end for
return H
```

The obvious advantage of this approach is that one starts out with a comparingly low dimensional parameter space and increases it iteratively, given that the current configuration allows for no further improvements. As a result of the iterative nature of such an algorithm, the execution time is proportional to the number of suboptimizations that have to be conducted.
Results of the final cost function value and the execution runtime are depicted in Table 3. In addition, we compare the obtained auxiliary system to the ones found with parallel tempering in Figure 7.

Table 3: Cost function values of the final solution of gradient-based optimization with iterative increase of the $\boldsymbol{H}$ matrix rank. In addition, execution runtime is given.

$\min(\mathcal{C}^2_{\text{grad,v}})$ ... Minimal cost function value of the gradient-based optimizer
$\text{avg}(\mathcal{C}^2_{\text{grad,v}})$ ... Average of the cost function value of the gradient-based optimizer
$\mathcal{C}^2_{\text{P.t.}}$ ... Cost function value of parallel tempering (comparison)
T ... Execution time

	$\min(\mathcal{C}^2_{\text{grad,v}})$	$\text{avg}(\mathcal{C}^2_{\text{grad,v}})$	$\mathcal{C}^2_{\text{P.t.}}$	T/s
2 Bath Sites	$192.1 \cdot 10^{-3}$	$192.2 \cdot 10^{-3}$	$192.1 \cdot 10^{-3}$	70 ± 3
4 Bath Sites	$53.17 \cdot 10^{-3}$	$58.66 \cdot 10^{-3}$	$53.18 \cdot 10^{-3}$	219 ± 39
6 Bath Sites	$0.674 \cdot 10^{-3}$	$0.704 \cdot 10^{-3}$	$0.683 \cdot 10^{-3}$	990 ± 284

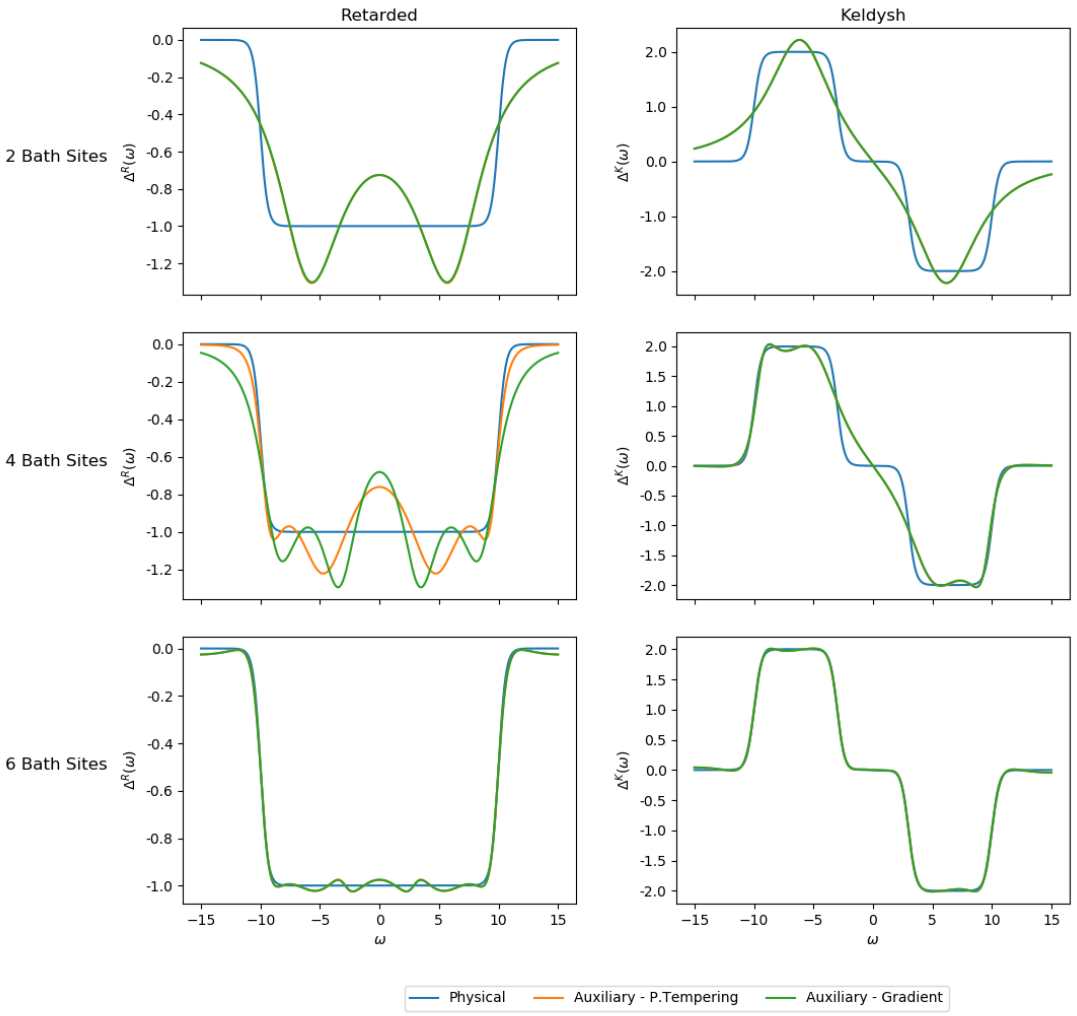

Figure 7: The solutions found by gradient-based iterative optimization compared to the original physical system and the parallel tempering solutions. Each row represents a different size of the auxiliary system . Left column: Retarded component of hybridization, Right column: Keldysh part.

6.5.1 Dependence on the Rank M

To illustrate the influence of the rank M of matrix $\boldsymbol{H}$ Figure 8 shows how the cost decreases as an additional vector $\boldsymbol{h}_i$ is added to $\boldsymbol{H}$. As a main result, it is observed that the rank increase in matrix $\boldsymbol{H}$ does not contribute essentially to the quality of the fit, especially when the auxiliary system has a high number of bath sites. For example, we considered an auxiliary system with **10 auxiliary bath sites**. In this case, we obtained the final cost value depending on the rank of $\boldsymbol{H}$ as given in Table 4. The graphical respresentation of those values given in Figure 9 exhibits virtually no improvement with $M > 2$. See also Figure 10 for the actual hybridization functions depending on M.

For practical purposes, for example quick tests, it may be sufficient to optimize only with $M = \lfloor \frac{N}{4} + 1 \rfloor$ (empiric suggestion), resulting in a dramatic improvement in execution speed. Furthermore, we can conclude that this parametrization combined with mentioned optimization procedure is able to perform equally in terms of final cost function value compared to parallel tempering while being faster in terms of execution runtime. Especially in the case of systems with more auxiliary bath sites one can profit from gradient-based optimization techniques. In particular, this method pushes the limit of how big an auxiliary system can be fitted in a feasible way.

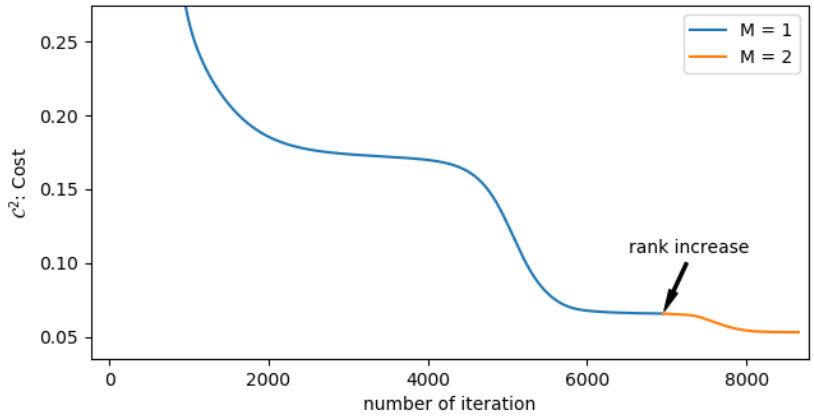

Figure 8: Improvement of the cost function as a the rank of $\boldsymbol{H}$ is increased. This example corresponds to an auxiliary system size of 4 bath sites.

Table 4: Cost function values at the increase points of the matrix rank M in the case of an auxiliary system with 10 auxiliary bath sites. The values are taken from a single run and thus only their relative values are representative.

M ... Matrix rank of $\boldsymbol{H}$
$\mathcal{C}^2$ ... Cost function value

M	$\mathcal{C}^2$
1	0.171
2	$6.456 \cdot 10^{-5}$
3	$3.329 \cdot 10^{-5}$
4	$3.317 \cdot 10^{-5}$
5	$3.323 \cdot 10^{-5}$

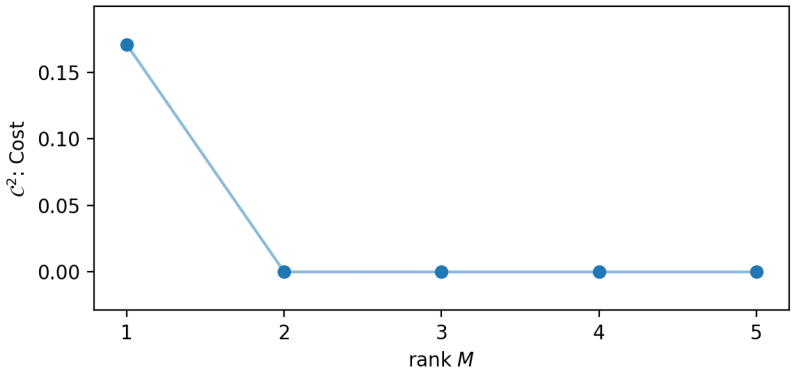

Figure 9: Visual representation of Table 4. Virtually no improvements are visible for $M > 2$. The considered system has 10 auxiliary bath sites.

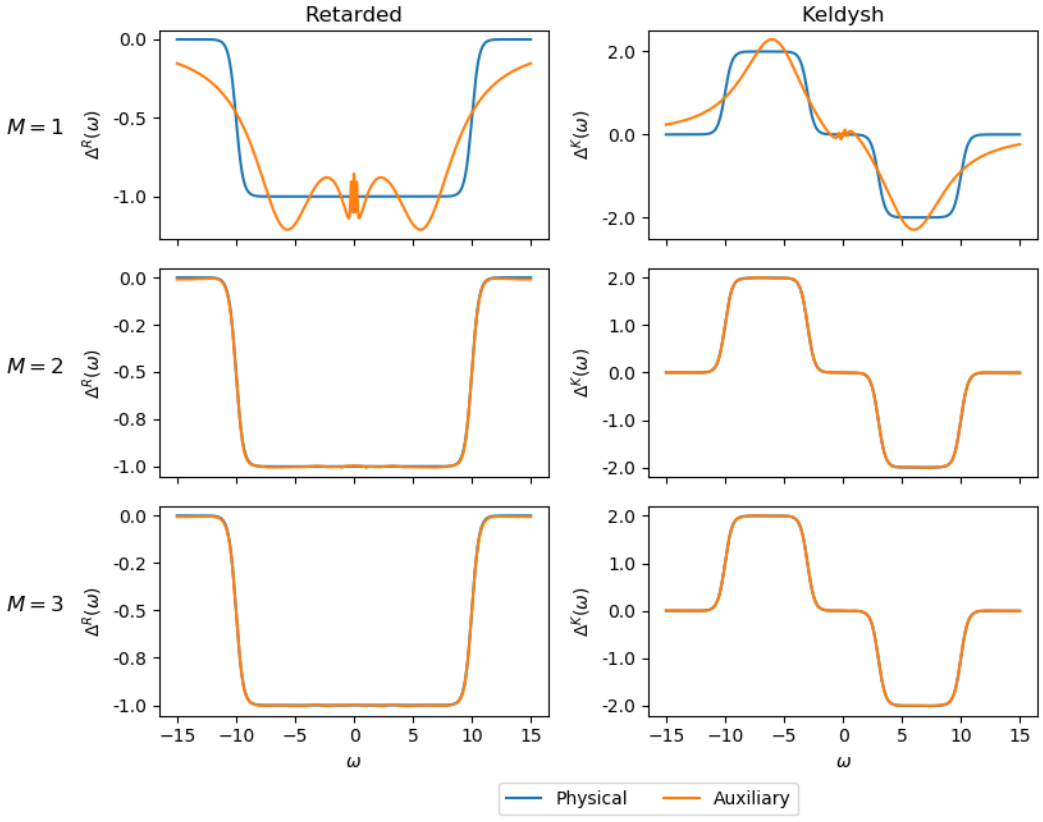

Figure 10: As M increases, the approximates gets better. However mark that already with $M = 2$ the system is very well fitted. This example considers a system with 10 auxiliary bath sites, i.e. the highest possible M is 5.

6.6 Runtime Comparison

To emphasize the runtime differences consider Figure 11. It shows how the different algorithms scale in runtime depending on the number of auxiliary bath sites. Although the gradient-based speedup in case of 4 bath sites is not drastic, the runtime improvement in case of 6 auxiliary bath sites is tenfold, an order of magnitude.

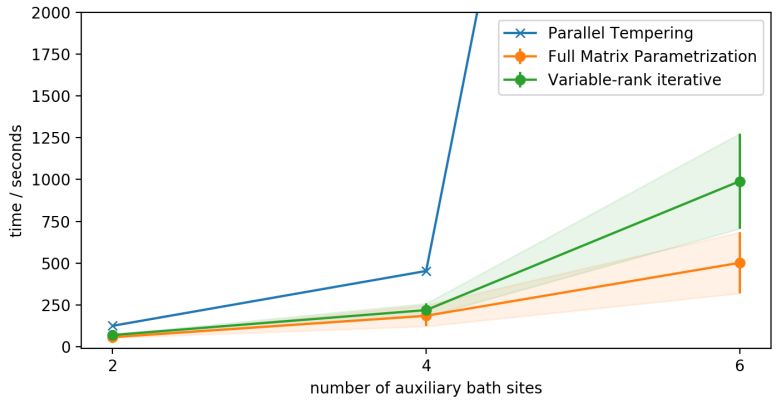

Figure 11: Runtime comparison of the different optimization algorithms. Keep in mind that the final value of the cost function is not identical. This plot is intented to visualize the scalability of gradient-based optimization to bigger auxiliary systems.

7 Conclusions

Out of equilibrium quantum systems are not only an interesting subject of study on its own but they are also salient for the development of the technologies. To be able study the properties of such a system the auxiliary master equation approach is a well established method. It is based on casting the complex systems into easier ones and thus one is posed with the challenge to find the right alternative approximate description. An optimization problem is obtained.

It was derived how such a parametrized auxiliary system can be fitted to a given target by use of first-order gradient information. Constraints on the matrices that are part of the systems description were taken into account and gave rise to parametrization schemes. As a result of the description of the auxiliary system, the derivation of the gradient is mostly concerned with the prominent chain rule.

Subsequently an appropriate gradient-based optimization algorithm was chosen, ADAM and was applied to the fitting problem under both parametrization schemes using state-of-the-art computer methods, `tensorflow` in particular.

It was shown that this work can be used either as an enhancement to the existing parallel tempering algorithm as the last stage in the optimization or, and presumably

more promising, as a standalone method to optimize the parameters of auxiliary systems with the variable-rank parametrization. In the latter case, the quality of the fit compares well to the pure parallel tempering solutions and the execution runtime is reduced by an **order of magnitude**.

References

[Abadi et al., 2016] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467.

[Arrigoni et al., 2013] Arrigoni, E., Knap, M., and von der Linden, W. (2013). Nonequilibrium dynamical mean-field theory: An auxiliary quantum master equation approach. *Phys. Rev. Lett.*, 110:086403.

[Bebendorf, 2008] Bebendorf, M. (2008). *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*. Springer Publishing Company, Incorporated, 1st edition.

[Breuer and Petruccione, 2007] Breuer, H. and Petruccione, F. (2007). *The Theory of Open Quantum Systems*. OUP Oxford.

[Dorda et al., 2014] Dorda, A., Nuss, M., von der Linden, W., and Arrigoni, E. (2014). Auxiliary master equation approach to nonequilibrium correlated impurities. 89(16):165105.

[Duchi et al., 2010] Duchi, J., Hazan, E., and Singer, Y. (2010). Adaptive subgradient methods for online learning and stochastic optimization.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.