Designing for Geometrical Symmetry Exploitation

André Yamba Yamba* Krister Åhlander*
Malin Ljungberg*

April 3, 2006

Abstract

Symmetry exploiting software based on the generalized Fourier transform (GFT) is presented from a practical design point of view. The algorithms and data structures map closely to the relevant mathematical abstractions, which primarily are based upon representation theory for groups. Particular care has been taken in the design of the data layout of the performance sensitive numerical data structures.

The use of a vanilla strategy is advocated for the design of flexible mathematical software libraries: An efficient general-purpose routine should be supplied, to obtain a practical and useful system, while the possibility to extend the library and replace the default routine with a special-purpose—even more optimized—routine should be supported.

Compared with a direct approach, the performance results show the superiority of the GFT-based approach for so-called dense equivariant systems. The GFT application is found to be well suited for parallelism.

1 Introduction

Many engineering problems in the real world exhibit a significant amount of geometrical symmetry. Just to mention a few examples, we note that groups of conductors used for high voltage electricity transport are often placed symmetrically in order to minimize disturbances, Buckminster balls are based on the symmetry of an icosahedron in order to maximize volume with a minimum of material, and propellers, used in technical applications ranging from turbines to airplanes, exhibit a cyclic symmetry.

In this paper, we are interested in the numerical simulation of partial differential equations (PDEs) that evolve in symmetrical regions, and we are concerned with the exploitation of this symmetry in the numerical algorithms. Our approach is based on the generalized Fourier transform (GFT), and has its origin in work by Allgower et al. [6] The approach can be particularly useful for a class of dense linear algebra systems that stem from geometrically symmetric domains. As explained below, systems in this class are referred to as *dense*

*Department of Information Technology, Uppsala University, Box 337, S-751 05 Uppsala, Sweden, anya9527@user.it.uu.se, Krister.Ahlander@it.uu.se, Malin.Ljungberg@it.uu.se

equivariant systems, and they may arise for instance when the boundary element method is used [14].

The GFT approach can also be useful for geometries that are almost symmetrical, or partly symmetrical. An almost symmetric problem can be approximated as symmetrical, and the solution of this problem can be used to derive a preconditioner [32]. Block-circulant preconditioners, see e.g. [19, 20], can be seen as a special case of this strategy. Another approach can be applied for partly symmetric problems. Here, it is possible to use domain decomposition methods and treat the symmetric part in isolation, thereby exploiting the symmetry [9].

However, our impression is that the GFT and its potential are relatively unknown in the scientific computing community. This is partly explained by the fact that the underlying mathematics, in particular representation theory for groups, is not common knowledge among scientific computing researchers. With adequate software support, however, we believe that usage of the GFT for symmetric problems could increase.

Software for group theory has a strong tradition with the group theory language Cayley from 1976 as a noteworthy landmark [12]. At present, "there are two systems which are particularly well suited for computations with groups: GAP and MAGMA" [28]. In the related research area of fast GFTs [26], GAP has been used e.g. by Egner and Püschel [13]. However, for the number-crunching applications that we are interested in, we think stand-alone software is more appropriate.

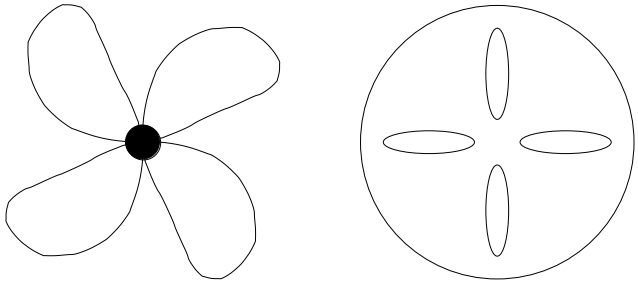
In this paper, we present a software package that is designed particularly with the solution of dense equivariant systems in mind. We are aware of two other packages dedicated to this application [15, 21]. The first of theses packages handles so called fix points, while the second focusses on a design based upon generic programming. In comparison with these projects, we have paid particular attention to the layout of the data structures that affect performance the most. We use LAPACK [7] for efficient matrix computations. The ANSI-C software abstractions are based closely on the mathematical abstractions, in order to achieve understandability and flexibility.

The outline is as follows. In Section 2, we give an overview of the mathematical machinery required for the GFT. In Sections 3 and 4, we outline how the software is designed and implemented, and we also discuss implementation variants, both general-purpose "vanilla" routines and special-purpose, fast routines. Section 5 shows performance results for the variants mentioned, and makes comparisons between a symmetry-exploiting solve and a direct solve of a dense equivariant system. These comparisons clearly show the importance of exploiting symmetries, as concluded in Section 6. Here, we also discuss the vanilla strategy for mathematical software.

2 Background

Consider the geometries of Figure 1. The propeller shape in (a) is clearly invariant under rotations by 90, 180, and 270 degrees, as well as under the identity transformation (a rotation by 0 or 360 degrees). The cylinder shape in (b) is invariant under the same rotations, as well as under four reflections. We notice the connection between symmetries and groups; the rotations in (a) form the cyclic group $\mathcal{C}_4$ with four elements whereas the rotations plus the reflections in

(b) form $\mathcal{D}_4$, the dihedral group with 8 elements. The transformations under which a symmetric shape is invariant is referred to as the shape's *symmetry group*.



(a) $\mathcal{C}_4$ symmetry (b) $\mathcal{D}_4$ symmetry

Figure 1: Geometries suitable for GFT methods.

Symmetric shapes and their corresponding symmetry groups are important in many applications. A linear operator $\mathcal{L}$, such as the Laplacian or an integral operator, is *equivariant* if it commutes with every transformation g in the symmetry group, i.e., $\mathcal{L}g = g\mathcal{L}$.

In this section, we give an overview of how a corresponding discrete operator is block-diagonalized by the GFT. The underlying mathematical theory is based on representation theory for groups. We summarize the theory in order to motivate our design and to make the paper self-contained. The notation is based on a recent introduction to the subject [3].

2.1 Groups and representations

First, we need some basic group theory. A *group* $\mathcal{G}$ is a set of elements that is closed under an associative binary operation, that has an identity element e, and for which every element g has an inverse g^{-1}. We point out that the groups we consider do not need to be abelian (commutative). The dihedral groups, for example, are nonabelian because a reflection followed by a rotation is not the same as a rotation followed by a reflection. We are concerned with finite groups, i.e., the number of elements $|\mathcal{G}| < \infty$. Actually, since we are interested in symmetric shapes in three dimensions, we are interested in relatively small groups. For the applications that we have in mind, the largest group of interest is the symmetry group of the icosahedron, which has 120 elements [5].

We consider groups *acting* on indices, thus partitioning the index set into *orbits*. An orbit of an index i consists of all indices ig obtained by letting all group elements g act on i. In this paper, we consider the case where all orbits have $|\mathcal{G}|$ elements, i.e., the case when the action is *free*.

Second, we need some representation theory. Given a group $\mathcal{G}$, a *complex representation* ρ of dimension d_ρ is a map $\rho : \mathcal{G} \to \mathbb{C}^{d_\rho \times d_\rho}$ such that

$$\rho(gh) = \rho(g)\rho(h), \ g, h \in \mathcal{G}. \tag{1}$$

We focus on complex representations here, but real representations are be defined analogously.

Two representations ρ and σ are *equivalent* if they are related via a similarity transform, i.e., $\rho(g) = T\sigma(g)T^{-1}$ for all $g \in \mathcal{G}$ and some nonsingular $T \in \mathbb{C}^{d_\rho \times d_\rho}$. A representation is *reducible* if it is equivalent to a block-diagonal representation. This means that ρ is reducible if there exists a nonsingular T such that

$$T\rho(g)T^{-1} = \begin{pmatrix} \sigma(g) & 0 \\ 0 & \tau(g) \end{pmatrix}, \; g \in \mathcal{G},$$

where the representations σ and τ have dimensions d_σ and d_τ that are strictly between 0 and d_ρ. An *irreducible* representation cannot be block-diagonalized as above. For example, any representation of dimension 1 is irreducible.

Every representation can be reduced into irreducible representations. For every finite group $\mathcal{G}$, there exists a complete list $\mathcal{R}$ of nonequivalent irreducible representations [29]. The list of representations can be understood as a way to represent the elements of a group $\mathcal{G}$ as block-diagonal matrices, a point of view that we have discussed elsewhere [4]. The number of nonequivalent irreducible representations (the number of blocks in the block-diagonal matrices) and the dimension d_ρ of each representation (the dimension of the corresponding block) are properties of the group in question. These properties also depend upon whether the representation is real or complex. For complex representations and finite groups, it holds that

$$\sum_{\rho \in \mathcal{R}} d_\rho^2 = |\mathcal{G}|. \tag{2}$$

Third, we introduce the generalized Fourier transform as an isomorphism between the group algebra and its corresponding Fourier algebra. The *group algebra* $\mathbb{C}\mathcal{G}$ is a vector space $\mathbb{C}^{|\mathcal{G}|}$ where the group elements are basis vectors and for $u \in \mathbb{C}\mathcal{G}$ we use the notation $u = \sum_{g \in \mathcal{G}} u(g)g$. The group algebra $\mathbb{C}\mathcal{G}$ is equipped with a convolution product $u, v \mapsto u * v$ where

$$(u * v)(g) = \sum_{h \in \mathcal{G}} u(h)v(h^{-1}g).$$

The corresponding Fourier algebra is a block-diagonal matrix algebra $\widehat{\mathbb{C}\mathcal{G}} = \bigoplus_{\rho \in \mathcal{R}} \mathbb{C}^{d_\rho \times d_\rho}$. The generalized Fourier transform (GFT) is a mapping $\mathbb{C}\mathcal{G} \to \widehat{\mathbb{C}\mathcal{G}}$ defined by

$$\hat{u}(\rho) = \sum_{g \in \mathcal{G}} u(g)\rho(g), \rho \in \mathcal{R}, \tag{3}$$

where $\hat{u} = \bigoplus_{\rho \in \mathcal{R}} \hat{u}(\rho)$. We remark that the GFT depends upon the choice of bases for the representations in $\mathcal{R}$. The inverse GFT (IGFT) is defined by

$$u(g) = \sum_{\rho \in \mathcal{R}} \frac{d_\rho}{|\mathcal{G}|} \text{trace}\left(\hat{u}(\rho)\rho(g^{-1})\right), g \in \mathcal{G}. \tag{4}$$

2.2 Exploiting equivariance

We will now discuss how the basic concepts introduced above can be used to design efficient algorithms for the kinds of problems that we are interested in solving. Thus, we consider geometries that are invariant under a symmetry group $\mathcal{G}$ of transformations, and we study the discretization matrix $\mathbf{A}$ of an

equivariant PDE operator. Provided that the discretization is symmetry respecting, $\mathbf{A}$ becomes equivariant [1]. In the context of matrices, this means that

$$\mathbf{A}_{i,j} = \mathbf{A}_{ig,jg}, \tag{5}$$

where i, j are discretization indices acted upon by $g \in \mathcal{G}$. Equivariant discretization matrices are encountered in the context of many different numerical methods, e.g., the finite emelement method [10, 1], finite differences [33], discretization via radial basis function [17, 18], or as we mentioned in the introduction, the boundary element method [6, 9].

In this paper, we assume that the discretization of the PDE leads to a dense equivariant linear system of equations $\mathbf{Ax} = \mathbf{b}$. Furthermore, we assume that the group action partitions the index set $\mathcal{I}$ of size n into m orbits of size $|\mathcal{G}|$. By picking one index from each orbit we may represent the orbits by a suitable selection $\mathcal{S} \subset \mathcal{I}$ [6].

An overview of the symmetry exploitation strategy is given in the commutative diagram of Figure 2. The diagram shows that an original matrix vector multiplication $\mathbf{Ax}$ in vector space $\mathbb{C}^n$, where $\mathbf{A}$ is equivariant, can be reformulated as a convolution $A*x$ in the vector space $\mathbb{C}^m\mathcal{G}$ or as a multiplication $\hat{A}\hat{x}$ in the vector space $\widehat{\mathbb{C}^m\mathcal{G}}$. As discussed in more detail below, we refer to $\mathbb{C}^m\mathcal{G}$ and $\widehat{\mathbb{C}^m\mathcal{G}}$ as the *group space* and the *Fourier space*, respectively. We point out that the key to exploiting symmetry and equivariance for our kind of applications is to do the expensive work in the appropriate space.

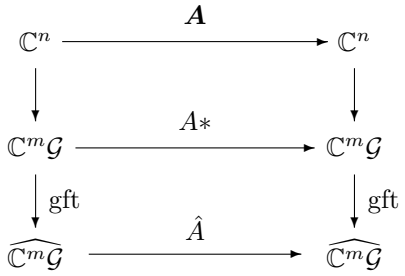


Figure 2: Relationships between the different spaces.

Group space formulation It is evident that the equivariance property (5) implies that $\mathbf{A}$ contains a number of duplicates. The group space formulation avoids this by the following invertible mappings, for $i, j \in \mathcal{S}$ and $g \in \mathcal{G}$.

$$\begin{aligned} A_{i,j}(g) &= \mathbf{A}_{ig,j}, \\ x_i(g) &= \mathbf{x}_{ig}, \\ b_i(g) &= \mathbf{b}_{ig}. \end{aligned}$$

Here, x and b belong to the vector space $\mathbb{C}^m\mathcal{G} = \mathbb{C}^m \otimes \mathbb{C}\mathcal{G}$. They contain the same data as their vector space counterparts $\mathbf{x}$ and $\mathbf{b}$. A difference is that while $\mathbf{x} \in \mathbb{C}^n$ is indexed with one index $i \in \mathcal{I}$, $x \in \mathbb{C}^m\mathcal{G}$ is indexed with two indices $i \in \mathcal{S}$ and $g \in \mathcal{G}$. Our notational convention is that $x_i \in \mathbb{C}\mathcal{G}$, $x(g) \in \mathbb{C}^m$, and $x_i(g) \in \mathbb{C}^m$.

Similarly, $A \in \mathbb{C}^{m\times m}\mathcal{G} = \mathbb{C}^{m\times m} \otimes \mathbb{C}\mathcal{G}$, and by the same notational convention we have that $A_{i,j} \in \mathbb{C}\mathcal{G}$ and $A(g) \in \mathbb{C}^{m\times m}$. The advantage with the above mappings is that duplicates in $\mathbf{A}$ are avoided, and A contains only a fraction $1/|\mathcal{G}|$ of the elements in $\mathbf{A}$. This is beneficial not only in terms of storage but also when the discrete operator itself is constructed, since duplicate elements need not be computed.

By generalizing the group algebra convolution, it is simple to show that the original linear system of equations can be expressed as $A * x = b$. The group space convolution $A * x$ is given by

$$(A * x)(g) = \sum_{h\in\mathcal{G}} A(h)x(h^{-1}g),$$

where a standard matrix vector product $\mathbb{C}^{m\times m} \times \mathbb{C}^m \to \mathbb{C}^m$ is used for the products $A(h)x(h^{-1}g)$.

We conclude that the original equation can be solved by solving its counterpart in group space instead. Note that x and b are vectors according to the vector space axioms, even though they are not the kind of vectors that computational scientists usually deal with. In the same spirit, we will refer to A as a *matrix* in group space, even though it is not a standard matrix.

Fourier space formulation In order to solve the problem in group space, it is reformulated as a linear system of equations in Fourier space. This is achieved by applying the GFT in the group algebra element-wise. The *vector GFT* is given by

$$\hat{x}_i(\rho) = \sum_{g\in\mathcal{G}} x_i(g)\rho(g), \ \rho \in \mathcal{R}, i \in \mathcal{S}. \tag{6}$$

Here, $\hat{x} \in \widehat{\mathbb{C}^m\mathcal{G}} = \mathbb{C}^m \otimes \widehat{\mathbb{C}\mathcal{G}}$ can be understood as a block-diagonal matrix where block ρ, i.e. $\hat{x}(\rho)$, has dimensions $md_\rho \times d_\rho$. The *matrix GFT* is given by

$$\hat{A}_{i,j}(\rho) = \sum_{g\in\mathcal{G}} A_{i,j}(g)\rho(g), \ \rho \in \mathcal{R}, i, j \in \mathcal{S}. \tag{7}$$

The matrix $\hat{A} \in \widehat{\mathbb{C}^{m\times m}\mathcal{G}} = \mathbb{C}^{m\times m} \otimes \widehat{\mathbb{C}\mathcal{G}}$ is a block-diagonal matrix where block ρ, i.e. $\hat{A}(\rho)$, is $md_\rho \times md_\rho$.

It is easy to show that $A * x = b$ corresponds to $\hat{A}\hat{x} = \hat{b}$, a matrix matrix equation where all matrices are block-diagonal. We refer to $\hat{A}$ as *the* matrix in Fourier space, while we refer to $\hat{x}$ and $\hat{b}$ as Fourier space vectors, even though they actually are block-diagonal matrices. In summary, the formulation in Fourier space is a block-diagonal version of the original linear systems of equations. It is much cheaper to solve the system in Fourier space, since each block is smaller than the original matrix and each block equation can be solved separately. For applications where dense equivariant systems are to be solved, this block-diagonalization is the most important way to exploit equivariance.

Regarding the GFT formulas (6) and (7), they are generic with respect to different groups and choices of irreducible representations. By studying the structure of the groups and their representations, it is possible to derive fast versions of the GFT, so called fast GFTs [23, 25]. The standard FFT is actually a prominent example of this. The FFT is a fast version of the DFT, which may

be interpreted as a GFT for a cyclic group. In Section 4, we describe the implementation both of general vanilla versions and of two fast GFT versions for $\mathcal{D}_4$.

3 Data structures

We strive to design data structures and algorithms that match the mathematical concepts, in order to get software which is easy to use and maintain. The UML [24] class diagram in Figure 3 is motivated by the theory in the previous section. It shows the static associations between the key concepts corresponding to the group $\mathcal{G}$, a list $\mathcal{R}$ of nonequivalent irreducible representations, and vectors and matrices in group space and Fourier space. The dynamic interactions among the participating objects are visible in the commutative diagram in Figure 2, and the overall GFT approach for solving an equivariant system $\mathbf{Ax} = \mathbf{b}$ is shown in Algorithm 1. Note that it is not necessary to form all of $\mathbf{A}$ and we therefore assume that the mapping into the group space equation $A * x = b$ has already been taken into account.

The design challenge, however, is to address computational efficiency and efficient memory handling as well. In the following subsections, we describe design considerations for the data structures for groups and irreducible representations, as well as vectors and matrices in group space and Fourier space. We refer to Yamba Yamba for a more detailed description [34].

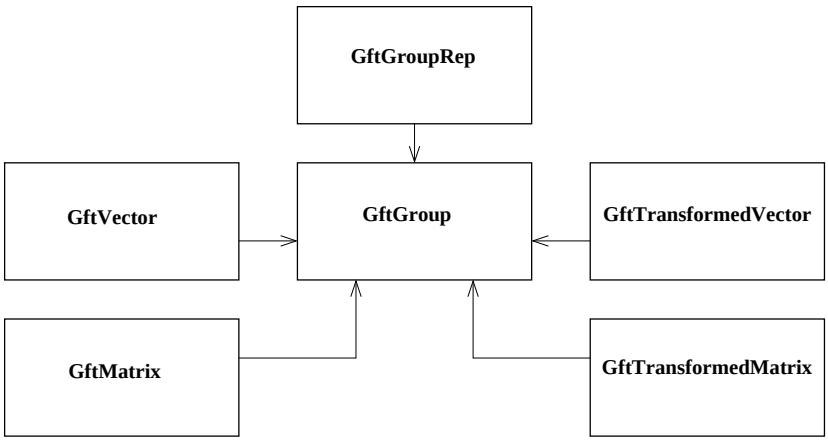


Figure 3: Class diagram for the key concepts **GftGroup**, **GftGroupRep**, **GftVector**, **GftMatrix**, **GftTransformedVector**, and **GftTransformed-Matrix**, representing $\mathcal{G}$, $\mathcal{R}$, $\mathbb{C}^m\mathcal{G}$, $\mathbb{C}^{m\times m}\mathcal{G}$, $\widehat{\mathbb{C}^m\mathcal{G}}$, and $\widehat{\mathbb{C}^{m\times m}\mathcal{G}}$, respectively.

3.1 Group

The concept of a group is clearly an important abstraction, which must supply the group order, the group operation, the inverse operation and so forth. There are different ways to implement groups, taking advantage of special properties of different groups or exploiting reuse between groups. A cyclic group $\mathcal{C}_N$, for example, can be represented as integers and the group operation can

Algorithm 1 GFT approach for solving an equivariant system $A * x = b$.

Let $\hat{A} = \text{gft}(A)$, $\hat{b} = \text{gft}(b)$
for $\rho \in \mathcal{R}$ **do**
  Solve $\hat{A}(\rho)\hat{x}(\rho) = \hat{b}(\rho)$
end for
Let $x = \text{igft}(x)$

be implemented as addition modulo N. Another structure that is simple to exploit is the direct product $\mathcal{G}_1 \times \mathcal{G}_2$, which is the Cartesian set of groups $\mathcal{G}$ and $\mathcal{H}$, and where the group operation is $(g_1, g_2)(h_1, h_2) = (g_1 h_1, g_2 h_2)$, for $(g_1, g_2), (h_1, h_2) \in \mathcal{G}_1 \times \mathcal{G}_2$.

Previously, we have discussed how C++ template mechanisms can be used to obtain a clean group interface and efficient variant implementations of different groups, since compile time polymorphism can be exploited [21]. In that design, cyclic groups are objects of the parametrized class `Cyclic<N>`, and the direct product of for example $\mathcal{C}_4$ with $\mathcal{C}_8$ is represented by the class

```
typedef DirectProduct< Cyclic<4>, Cyclic<8> > C4timesC8 ;
```

In the present project, we have chosen to implement in C and to keep the design at a low abstraction level. In the spirit of the vanilla design philosophy, we have chosen to implement the group by representing each group element by an integer, and to supply the group operation and the inverse operation as arrays of integers. In our context, there are three issues that we want to emphasize:

1. The implementation of the group is not time-critical. The important task is to design a useful API, and to have a working implementation. If a future application requires a more sophisticated solution, it is possible to supply special-purpose implementations for special groups.

2. We focus on symmetry groups of geometries in 2D and 3D. The most complicated group for the applications that we have in mind is the symmetry group of the icosahedron which only has 120 elements [5]. It is therefore not a serious restriction to use arrays for maintaining the group operation.

3. Our implementation allows a user to input the group operation and inverse operation as tables from a file, generated by hand or perhaps by another program. Since this solution is rather error-prone, we provide a routine which checks that the group axioms hold.

A group is represented by the data structure **GftGroup**, see the UML diagram in Figure 4 and the description of the application program interface (API) given below. Note that it is simple to implement variations of **GftGroup** data structure, for instance to support cyclic groups without having tables for the group operation. We also stress that the number of irreducible representations and their dimensions depend upon the underlying field, supplied by the user.

Group API

The operations of **GftGroup** are:

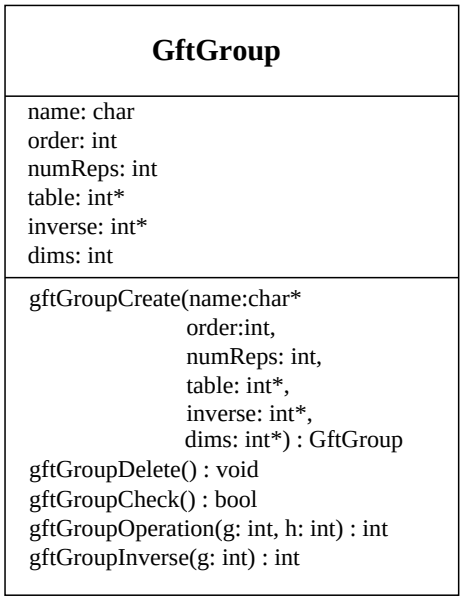


Figure 4: GftGroup, implementing $\mathcal{G}$.

gftGroupCreate Constructor. Create a group by supplying the requested inparameters.

gftGroupDelete Destructor.

gftGroupCheck Check that the group axioms (associativity, identity, inverse) hold.

gftGroupOperation Return the group operation gh.

gftGroupInverse Return the inverse element g^{-1}.

We also supply get operations for the name, the order, the number of irreducible representations and their dimensions. The data members of **GftGroup** are:

name String containing the name of the group.

order Positive integer containing the group order.

num_reps Positive integer containing the number of irreducible representations.

table $order \times order$ matrix containing the group operation table.

inverse Array containing the inverses.

dims Array containing the dimensions of the irreducible representations.

3.2 Irreducible Representations

The list $\mathcal{R}$ of nonequivalent irreducible representations is maintained by the **GftGroupRep** data structure. There are several more or less sophisticated ways of implementing this important abstraction. Here, we focus on the interface and we provide a vanilla implementation, in analogy with **GftGroup**.

Let us first formalize the notation in more detail. Let q be the number of representations in $\mathcal{R}$, and enumerate the nonequivalent irreducible representations from $\mathcal{R}_0$ to $\mathcal{R}_{q-1}$. We discuss complex representations here, and the r'th irreducible representation is thus $\mathcal{R}_r : \mathcal{G} \to \mathbb{C}^{d_r \times d_r}$, where $d_r = \dim \mathcal{R}_r$. For a given representation $\mathcal{R}_r$ and a given group element g, we denote an element in the matrix $\mathcal{R}_r(g)$ by $\mathcal{R}_r(g)_{\mu,\nu}$ where $\mu, \nu \in [0, \ldots, d_r - 1]$. Equation (2) shows that $\mathcal{R}$ contains $|\mathcal{G}|$ elements $\mathcal{R}_r(g)_{\mu,\nu}$ in total.

In our vanilla implementation of **GftGroupRep**, all $|\mathcal{G}|^2$ elements of $\mathcal{R}$ are organized in a straight-forward array. In order to obtain fast access to individual representations $\mathcal{R}_r$, the $d_r^2|\mathcal{G}|$ elements of $\mathcal{R}_r$ are stored consecutively. To determine an appropriate ordering of the remaining indices $\mu, \nu \in [0, \ldots, d_r - 1]$ and $g \in \mathcal{G}$, we examine how $\mathcal{R}_r$ is accessed during the GFT (3). The transform $x \mapsto \hat{x}$ is carried out as a sum over $g \in \mathcal{G}$. Therefore, we store $\mathcal{R}_r(\cdot)_{\mu,\nu}$, i.e., the $|\mathcal{G}|$ elements $\mathcal{R}_r(g)_{\mu,\nu}, g \in \mathcal{G}$, consecutively. As a consequence, we notice that the transform (3) actually can be expressed as a matrix vector multiplication. Let each row of a $|\mathcal{G}| \times |\mathcal{G}|$ matrix $\mathbf{R}$ contain $\mathcal{R}_r(\cdot)_{\mu,\nu}$, and let $\mathrm{col}(\hat{x})$ denote a "columnization" of $\hat{u}$, i.e., organize the elements in the block-diagonal matrix $\hat{u} \in \widehat{\mathbb{C}\mathcal{G}}$ as a column vector with $|\mathcal{G}|$ elements. Then we see that

$$\mathrm{col}(\hat{u}) = \mathbf{R}u \tag{8}$$

computes the GFT, provided that the indexing of $\mathbf{R}$ matches the indexing of u and $\hat{u}$.

Thus, by storing elements of $\mathcal{R}$ as a $|\mathcal{G}| \times |\mathcal{G}|$ matrix $\mathbf{R}$, the GFT can be achieved by a call to LAPACK. In addition, we see that a convenient way to implement a vanilla version of the IGFT (4) is by computing $\mathbf{R}^{-1}$ by another LAPACK call, and carry out the matrix vector multiplication $u = \mathbf{R}^{-1}\mathrm{col}(\hat{u})$.

The data structure **GftGroupRep** is illustrated in Figure 5 and described below. Notice that the number of irreducible representations as well as their respective dimensions are available through the **GftGroup** member. Regarding precision, both single precision complex and double precision complex are supported via a link time option. We also support single precision and double precision real numbers. This means that the vanilla implementation supports real GFTs for groups where the dimensions of the real and complex nonequivalent irreducible representations coincide. This is the case for many important groups, e.g., all dihedral groups, the symmetry group of the cube, and the symmetry group of the icosahedron [22].

Group Representation API

The operations of **GftGroupRep** are:

gftGroupRepCreate Constructor. Create a list of nonequivalent irreducible representations by supplying the requested inparameters.

gftGroupRepDelete Destructor.

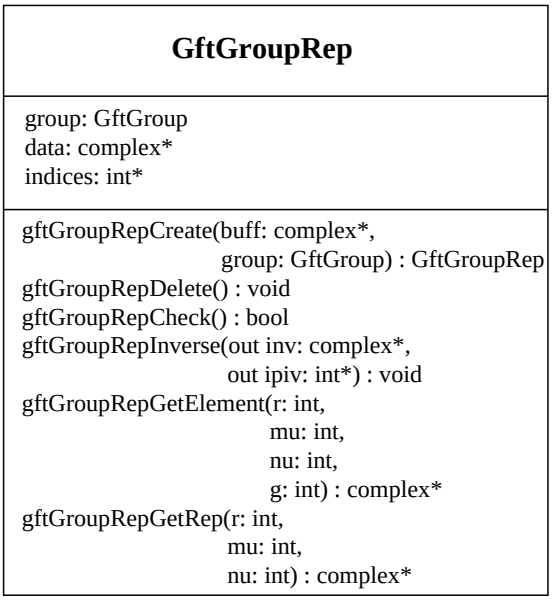


Figure 5: GftGroupRep, implementing $\mathcal{R}$.

gftGroupRepCheck Check that each representation $\mathcal{R}_r$ is a representation, i.e., check that $\mathcal{R}_r(gh) = \mathcal{R}_r(g)\mathcal{R}_r(h)$ for all $g, h \in \mathcal{G}$ and all $\mathcal{R}_r \in \mathcal{R}$.

gftGroupRepInverse Return $\mathbf{R}^{-1}$, obtained via a Lapack call. The out-parameter `ipiv` contains pivot information.

gftGroupRepGetElement Return $\mathcal{R}_r(g)_{\mu,\nu}$.

gftGroupRepGetRep Return $\mathcal{R}_r(\cdot)_{\mu,\nu}$.

The data members of **GftGroupRep** are:

group An association to the corresponding group.

data The matrix $\mathbf{R}$, containing all elements of $\mathcal{R}$.

blockIndices An array containing the array indices of the first element of each of the different irreducible representations in the data buffer. This array is maintained in order to facilitate a fast computation of the addresses of the elements of $\mathcal{R}$.

3.3 Group space vectors and matrices

When transforming matrices and vectors of an equivariant system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ and $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{C}^n$, into their corresponding elements in the group space, $A \in \mathbb{C}^{m\times m}\mathcal{G}$ and $x, b \in \mathbb{C}^m\mathcal{G}$, the computational effort can be minimized through careful selection of the numbering system used for the indices. Different numberings lead to different but equivalent equivariant systems. The transformation into the group space can be simplified by the selection of a numbering scheme which corresponds to the order in which the group elements are listed.

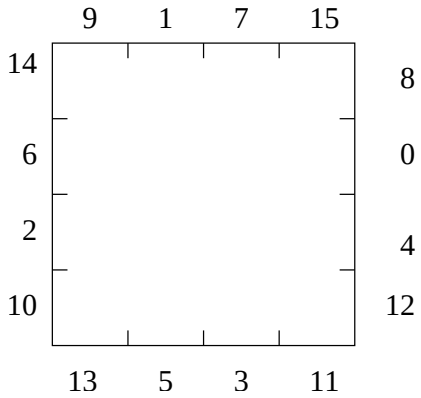


Figure 6: Group space indexing for $\mathcal{D}_4$ with $m = 2$.

With such a selection, the transformation of a vector into its group space counterpart is reduced to a reinterpretation of the contents of the data structures, cf. the discussion in connection with Algorithm 1.

Let $\mathcal{G}$ be the group describing the symmetry, let m be the number of orbits of the discretization. Under a free action of $\mathcal{G}$ a symmetry respecting discretization of m orbits has a number of elements equal to $n = m\,|\mathcal{G}|$. For all orbits, let $i \in 0, \ldots, m-1$ be the orbit index. Assign the subset of element indices $\{i\,|\mathcal{G}|, \ldots, (i+1)\,|\mathcal{G}| - 1\}$ to the elements of orbit i according to the following procedure. Pick o_i as the first element of the orbit, i.e., let o_i correspond to index $i\,|\mathcal{G}|$, and let g_k for $k = 0, \ldots, |\mathcal{G}|-1$ denote group element of index k, and assign to $o_i g_k$ index $i\,|\mathcal{G}| + k$. Figure 6 illustrates this numbering for a symmetry respecting discretization with two orbits for the case of group $\mathcal{D}_4$ generated by a rotation a and a reflection b. The group elements are enumerated in the order $e, a, a^2, a^3, b, ab, a^2b, a^3b$. Element 0 is chosen as the first elements of orbit $\{0, \ldots, 7\}$ and element 8 is chosen as the first element of orbit $\{8, \ldots, 15\}$. With this numbering the vector x is stored in the following order:

$$x_0(0) \ldots x_0(\gamma), x_1(0) \ldots x_1(\gamma), \ldots x_{m-1}(0) \ldots x_{m-1}(\gamma),$$

where $\gamma = |\mathcal{G}| - 1$.

A group space vector x is represented by the data structure **GftVector** and a group space matrix A is represented by the data structure **GftMatrix**. See Figures 7 and 8 and the API descriptions below. Notice that the original vector $\mathbf{x}$ can be represented by the same data structure as x, thanks to our choice of index ordering.

Group space vector API

The operations of **GftVector** are:

gftVectorCreate Constructor. Create a group space vector by supplying the requested inparameters. Typically used for initializing b.

gftVectorAlloc A second constructor, which omits initialization of data. Typically used for allocating x.

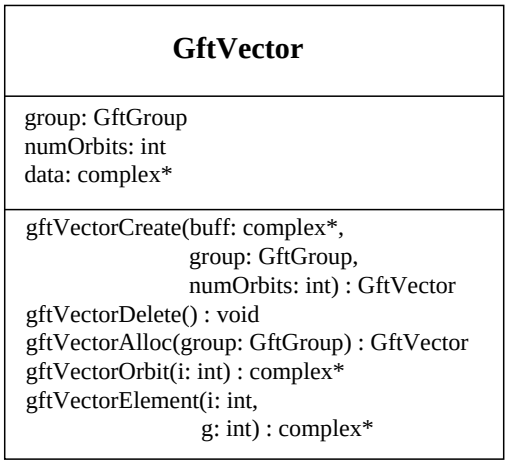


Figure 7: GftVector, implementing $\mathbb{C}^m\mathcal{G}$.

gftVectorDelete Destructor.

gftVectorOrbit Return address of x_i.

gftVectorElement Return address of element $x_i(g)$.

The data members of **GftVector** are:

group An association to the corresponding group.

num orbits The number of orbits.

data The elements of x.

Group space matrix API

The operations of **GftMatrix** are:

gftMatrixCreate Constructor. Create a group space matrix by supplying the requested inparameters.

gftVectorDelete Destructor.

gftMatrixOrbit Return address of $A_{i,j}$.

gftVectorElement Return address of element $A_{i,j}(g)$.

The data members of **GftMatrix** are:

group An association to the corresponding group.

num orbits The number of orbits.

data The elements of A.

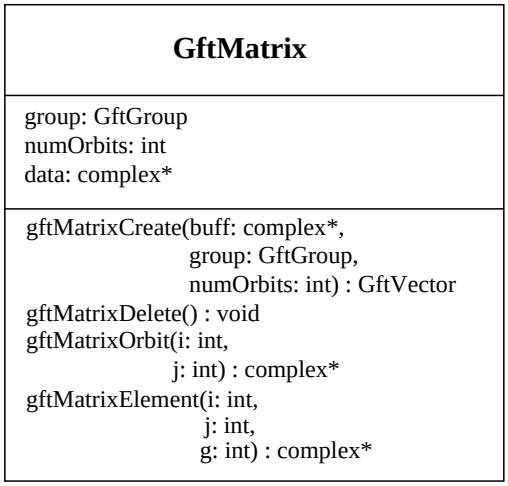


Figure 8: GftMatrix, implementing $\mathbb{C}^{m\times m}\mathcal{G}$.

3.4 Fourier space vectors and matrices

The data types representing elements in the Fourier spaces $\widehat{\mathbb{C}^m\mathcal{G}}$ and $\widehat{\mathbb{C}^{m\times m}\mathcal{G}}$ were designed with the primary goals of achieving efficient solution of the system $\widehat{A}\widehat{x} = \widehat{b}$ and to obtain efficient computation of the GFTs. To this end, we organized the data structures as follows.

Since the transformed equation $\hat{A}\hat{x} = \hat{b}$ actually consists of q independent equations $\hat{A}(\mathcal{R}_r)\hat{x}(\mathcal{R}_r) = \hat{b}(\mathcal{R}_r)$, for $r = 0, \ldots, q-1$, it is essential to store the elements of $\hat{A}(\mathcal{R}_r)$, $\hat{x}(\mathcal{R}_r)$, and $\hat{b}(\mathcal{R}_r)$ consecutively. If representation $\mathcal{R}_r$ has dimension $d_r > 1$, $\hat{x}(\mathcal{R}_r)$ and $\hat{b}(\mathcal{R}_r)$ are matrices with more than one column. Columns are stored consecutively, indexed by $\nu = 0, \ldots, d_r - 1$. The data layout of the Fourier space vectors is arranged using the following order of the indices, where r is the outermost index:

$$r = 0, \ldots, q-1;\ \nu = 0, \ldots, d_r - 1;\ \mu = 0, \ldots, d_r - 1;\ i = 0, \ldots, m-1.$$

The choice of index ordering for the vectors determines the ordering of the Fourier space matrices. Fourier space matrices are organized with the indices in the following order:

$$r = 0, \ldots, q-1;\ \nu = 0, \ldots, d_r - 1;\ j = 0, \ldots, m-1;\ \mu = 0, \ldots, d_r - 1;\ i = 0, \ldots, m-1.$$

This data layout allows us to solve $\hat{A}\hat{x} = \hat{b}$ via standard Lapack calls. Other layouts are possible, but it is essential to store blocks separately and it is important that the matrix layout and the vector layout match each other.

A Fourier space vector $\hat{x}$ is represented by the data structure **GftTransformedVector**, see Figure 9 and the API description below. A Fourier space matrix $\hat{A}$ is represented by the data structure **GftTransformedMatrix**, see Figure 10. The API is not further described, since operations and data members are almost identical to those for Fourier space vectors.

Fourier space vector API

The operations of **GftTransformedVector** are:

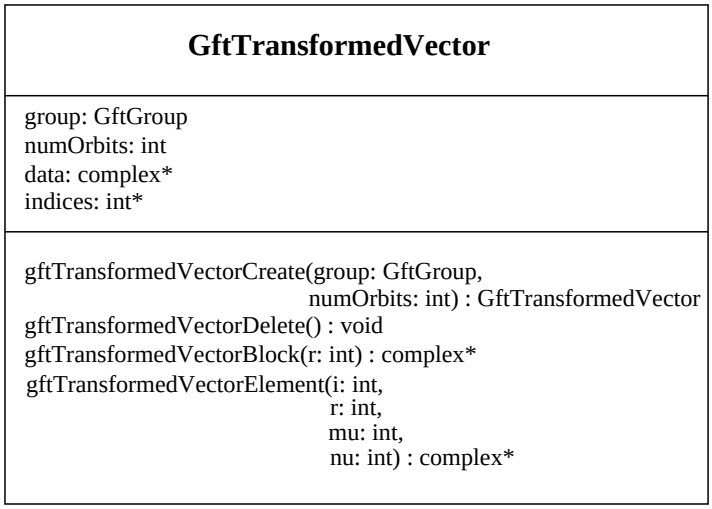


Figure 9: GftTransformedVector, implementing $\widehat{\mathbb{C}^m\mathcal{G}}$.

gftTransformedVectorCreate Constructor. Create a Fourier space vector by supplying the requested inparameters.

gftTransformedVectorDelete Destructor.

gftTransformedVectorBlock Return address of $\hat{x}(\mathcal{R}_r)$.

gftTransformedVectorElement Return address of element $\hat{x}_i(\mathcal{R}_r)_{\mu,\nu}$.

The data members of **GftTransformedVector** are:

group An association to the corresponding group.

num orbits The number of orbits.

data The elements of $\hat{x}$.

blockIndices An array that maintains the first index of each block.

4 Algorithms

Using the data structures described in the previous section, we have developed several implementations of the GFT algorithm. The purpose is to study how different strategies perform in the context of solving dense equivariant linear systems of equations with the GFT approach.

The first variants, **Vanilla**, are general and map closely to (6) and (7). The second **Matrix Multiplication** variant is based upon the notion of computing the GFT via a matrix vector multiplication (8). This makes it possible to use LAPACK. Finally we have implemented **Fast** variants for the case of $\mathcal{D}_4$. Here, the structure of the specific group and its list of nonequivalent irreducible representations are used to reduce the number of arithmetic operations in the GFT.

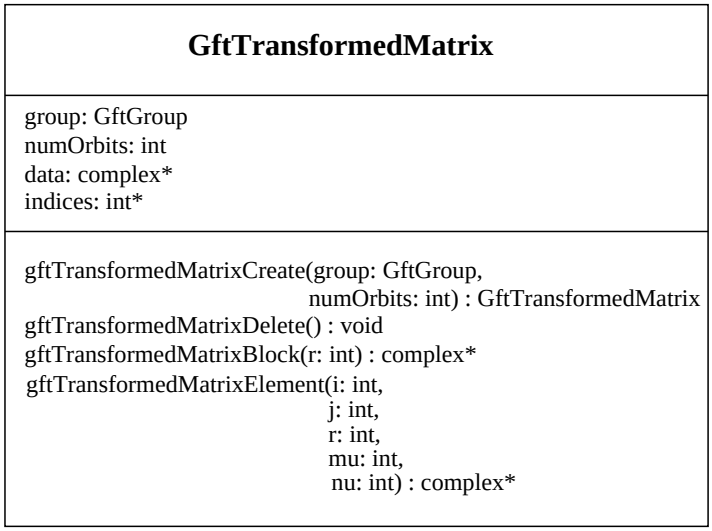


Figure 10: GftTransformedMatrix, implementing $\widehat{\mathbb{C}^{m\times m}\mathcal{G}}$.

4.1 Vanilla variants

The general **Vanilla** implementations map closely to (6) and (7). Six nested loops are used for the matrix GFT $\mathbb{C}^{m\times m}\mathcal{G} \stackrel{\text{gft}}{\rightarrow} \widehat{\mathbb{C}^{m\times m}\mathcal{G}}$, and five loops are used for the vector GFT $\mathbb{C}^{m}\mathcal{G} \stackrel{\text{gft}}{\rightarrow} \widehat{\mathbb{C}^{m}\mathcal{G}}$. Apart from roundoff errors, the ordering of the loops has no implications for the final result, but the performance of the algorithm depends on the loop ordering [16].

We have implemented two loop orderings, shown for the case of the matrix GFT in Algorithms 2 and 3. The first ordering is chosen to match the access pattern of $\hat{A}$, whereas the second ordering accesses the elements of A more efficiently. The computational complexity of both algoritms is $|\mathcal{G}|^2$ multiplications and $|\mathcal{G}|^2$ additions per group algebra GFT $\mathbb{C}\mathcal{G} \rightarrow \widehat{\mathbb{C}\mathcal{G}}$.

Algorithm 2 Vanilla 1 algorithm for the matrix GFT.

for $r = 0, q-1$ **do**
  for $\nu = 0, d_r - 1$ **do**
    for $j = 0, m-1$ **do**
      for $\mu = 0, d_r - 1$ **do**
        for $i = 0, m-1$ **do**
          $\hat{A}_{i,j}(\mathcal{R}_r)_{\mu,\nu} = 0$
          for $g \in \mathcal{G}$ **do**
            $\hat{A}_{i,j}(\mathcal{R}_r)_{\mu,\nu} = \hat{A}_{i,j}(\mathcal{R}_r)_{\mu,\nu} + \mathcal{R}_r(g)_{\mu,\nu} A_{i,j}(g)$
          end for
        end for
      end for
    end for
  end for
end for

Algorithm 3 Vanilla 2 algorithm for the matrix GFT.

for $j = 0, m-1$ **do**
  for $i = 0, m-1$ **do**
    for $r = 0, q-1$ **do**
      for $\nu = 0, d_r - 1$ **do**
        for $\mu = 0, d_r - 1$ **do**
          $\hat{A}_{i,j}(\mathcal{R}_r)_{\mu,\nu} = 0$
          for $g \in \mathcal{G}$ **do**
            $\hat{A}_{i,j}(\mathcal{R}_r)_{\mu,\nu} = \hat{A}_{i,j}(\mathcal{R}_r)_{\mu,\nu} + \mathcal{R}_r(g)_{\mu,\nu} A_{i,j}(g)$
          end for
        end for
      end for
    end for
  end for
end for

4.2 Matrix multiplication variant for the matrix GFT

Since the GFT can be understood as a matrix vector multiplication, we developed a version where the GFTs are carried out via calls to LAPACK. The matrix GFT is shown in Algorithm 4. The layout of A makes it possible to interpret A as a $\mathcal{G} \times m^2$ matrix, and the GFT is carried out by a single matrix matrix multiplication. Since the result must be permuted in order to fit the prescribed layout of $\hat{A}$, this variant uses a temporary memory buffer T.

Algorithm 4 Matrix multiplication matrix GFT.

Interpret A as a $\mathcal{G} \times m^2$ matrix
Compute $T = \mathbf{R}A$
Insert T into col($\hat{A}$)

4.3 Fast variants

By exploiting the structure of $\mathcal{R}$, fast GFTs can be developed. It is beyond the scope of this paper to discuss fast GFTs in detail, and we refer to Maslen and Rockmore for a survey [23, 25]. Fast GFTs for dihedral groups have been studied by e.g. Stiller [30]. We are interested in studying the importance of fast GFTs for our kind of applications, and we have developed two fast versions for the case of $\mathcal{D}_4$, where the GFT corresponds to multiplication with

$$\mathbf{R} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \end{pmatrix}.$$

In the case of $\mathcal{D}_4$, the structure of $\mathbf{R}$ is simple enough to allow development of fast GFTs by inspection. For the conventional vanilla approach, the application of the GFT requires $2|\mathcal{D}_4|^2 = 128$ arithmetic operations. The **Fast 1** variant $\widehat{u} = \text{fgft}_1(u)$ is computed by an algorithm that requires 12 temporary variables and 20 arithmetic operations per orbit. The algorithm avoids unnecessary multiplications and reuses temporary results, in a straight-forward fashion. The **Fast 2** variant reuses temporary results more agressively, and computes $\widehat{u} = \text{fgft}_2(u)$ with an algorithm that requires 6 temporary variables and 17 arithmetic operations per orbit. The details of the **Fast** algorithms are accounted for in Appendix A. The matrix GFT is carried out as shown in Algorithm 5, and the vector GFT is carried out analogously.

Algorithm 5 Fast matrix GFT variants $v = 1, 2$ for $\mathcal{D}_4$.

```
for j = 0, m − 1 do
  for i = 0, m − 1 do
    Â_{i,j} = fgft_v(A_{i,j})
  end for
end for
```

5 Results

In order to evaluate our implementations we have carried out several numerical experiments. These tests were done on a Sun Fire 15k server using the Sun WorkShop 6.2 compiler. Compiler flags were `-fast -xarch=v8plusb -dalign -xc99=%all -xO3`. The reported timings are obtained as the fastest result of ten consecutive runs, in order to avoid random effects. We have chosen maximum problem sizes in the order of 10000 unknowns, which is quite large on this platform.

The first study focuses on the serial performance of the GFT operation. Particularly, we are interested in comparing the flexible **Vanilla** implementations with the **Fast** versions. The second study examines the use of the GFT for solving dense equivariant systems. Third, we investigate the parallel performance of our GFT implementation. The group $\mathcal{D}_4$ is used for all performance results, but the **Vanilla** algorithms have also been tested for other groups.

5.1 Different implementations of the generalized Fourier transform

Figure 11 shows the serial performance of the different implementations described in Section 4. The best performing version is **Fast 1** and the worst is **Vanilla 1**. We note that even though **Fast 1** uses more arithmetic operations than **Fast 2**, it still performs better. We believe that the reason for this is that the aggressive reuse of temporary variables in the **Fast 2** algorithm, see Appendix A, makes it impossible for the superscalar processor to pipeline the computations [31].

The bad performance of the **Matrix Multiplication** variant indicates that the cost of using intermediate, temporary storage is high. An interesting result

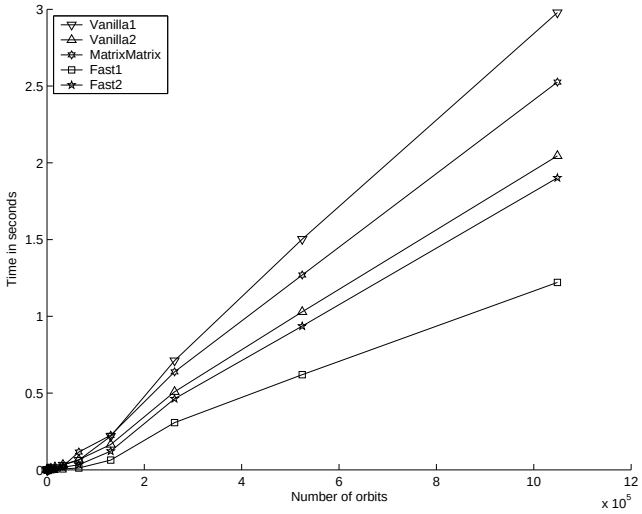


Figure 11: Comparison between the different GFT implementations.

is that **Vanilla 2** variant performs quite well. This general purpose variants perform almost as well as the **Fast 2** group specific implementation.

5.2 Solving equivariant systems

The execution time for solving a $\mathcal{D}_4$-equivariant system $\mathbf{Ax} = \mathbf{b}$ using the GFT, was compared to that of direct solution of the system. For each $m \in \{64, 128, 256, 512, 1024\}$ a randomly generated equivariant system $\mathbf{Ax} = \mathbf{b}$, with $\mathbf{A} \in \mathbb{C}^{n\times n}$, $n = 8m$, was solved using the symmetry exploiting method on one hand and direct solution on the other. For the symmetry exploiting case, the **Fast 1** algorithm was used for the GFT. We have measured the time taken to compute $\widehat{A} = \text{gft}(A)$, $\widehat{b} = \text{gft}(b)$, and $x = \text{igft}(\widehat{x})$ and for solving $\widehat{A}\widehat{x} = \widehat{b}$, and we compare the sum of these times to the time used to solve $\mathbf{Ax} = \mathbf{b}$ directly. In both cases LAPACK routines were used when solving the systems. Figure 12 shows a comparison between the times used by the two methods. The time complexity is actually $\mathcal{O}(n^3)$ in both cases, but the constant is much lower for the GFT approach.

The $\mathcal{O}(n^3)$ growth of the GFT-based approach is seen more clearly in Figure 13, where the time taken by each step of the symmetry exploiting method is shown. The most time consuming part of the GFT method is the solution of the transformed, block diagonal system. Compared to this time, the times for the GFT operations are almost insignificant. In particular the GFT of b and the IGFT of $\hat{x}$ take approximately 0.005% of the total time, and can therefore not be distinguished in the graph.

5.3 Parallelism and speedup

The performance of an OpenMP parallelization of the GFT approach was also examined. The test program was set to perform in parallel the computation of $\text{gft}(A) \in \mathbb{C}^{m\times m}\mathcal{G}$ with $m = 3072$ orbits and $\mathcal{G} = \mathcal{D}_4$. This corresponds to

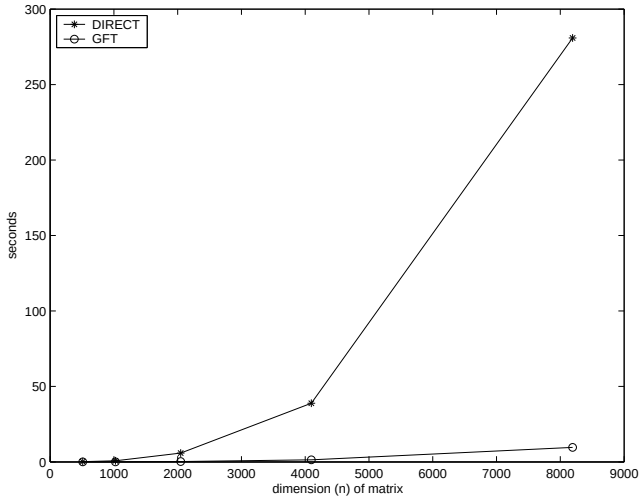


Figure 12: Comparison between direct and symmetry-exploiting solutions for different matrix sizes.

an $n \times n$ matrix with $n = 24576$. We computed $\widehat{A} = \mathrm{gft}(A)$, using $1, 2, 4, 8, 16$ and 24 CPUs. Figure 14 shows the relative speedup, calculated by dividing the time for one CPU by each of the times measured. The results show that this application is well suited for shared memory parallelism, and we think that the reason for this is that the GFT of different orbits are independent.

6 Conclusions

We have presented the design of a GFT-based symmetry-exploiting software. The basic idea is to transform a dense equivariant system of equations to a corresponding system in Fourier space, which can be solved much cheaper. Major design goals have been to facilitate efficient equation solving in Fourier space and to simplify the various mappings involved, including the GFT. Our results show that the GFT-based approach is very efficient compared to a direct approach. We have also shown that the matrix GFT is well suited for shared memory parallelism.

We would like to explicitly draw the attention to our design philosophy, since it has a broader applicability than just symmetry exploitation. Most importantly, our software is based closely upon the mathematical abstractions, which makes the software flexible and easy to understand [2]. We have verified the flexibility by extending the software to support other groups as well. In this context, we stress the role of the "vanilla strategy." By supplying a general-purpose GFT routine, a useful software is obtained. If, however, the vanilla version is not fast enough, it is possible to add special-purpose, fast routines. It is well-known that it is much easier to verify an optimized routine once a first version is in place [27]. Our point, however, is that the vanilla routine supplied by the framework should be optimized as far as possible while still being general.

Regarding the interesting topic of fast GFTs, we would like to make two observations. First, for the applications we are interested in, the most time

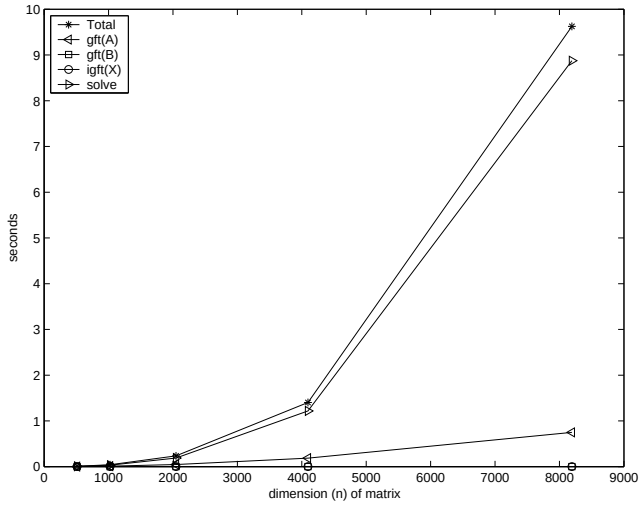


Figure 13: Time consumption of the different steps of the GFT-based approach.

consuming part is the system solving in Fourier space, and therefore we think that an optimized vanilla version is often good enough. Second, an interesting point is noted in Section 5.1. The performance of the **Fast 1** and **Fast 2** illustrate the fact that it is not only the number of arithmetical operations that is important. The underlying computer architecture, memory accesses, possibility of pipe-lining and so forth are also important factors that determine the performance of an algorithm. It would be an interesting challenge to develop fast GFT algorithms that take issues such as these into account.

For future work, we will address more groups and applications. One application that we have studied is the Black–Scholes equations [8] discretized with radial basis functions [11] on a square. By suitable transformations, the equations are transformed into the heat equation, leading to a dense $\mathcal{D}_4$-equivariant space discretization matrix. Even though time discretization and boundary treatment destroy equivariance, techniques similar to those presented in [9] can be used to obtain promising results [17, 18].

Acknowledgements

We thank Michael Thuné and Elisabeth Larsson for their valuable comments on the manuscript.

References

[1] K. Åhlander. Sparse Generalized Fourier Transforms. Technical Report 2005-043, Department of Information Technology, Uppsala University, 2005.

[2] K. Åhlander, M. Haveraaen, and H. Munthe-Kaas. On the role of mathematical abstractions for scientific computing. In R. Boisvert and P. Tang,

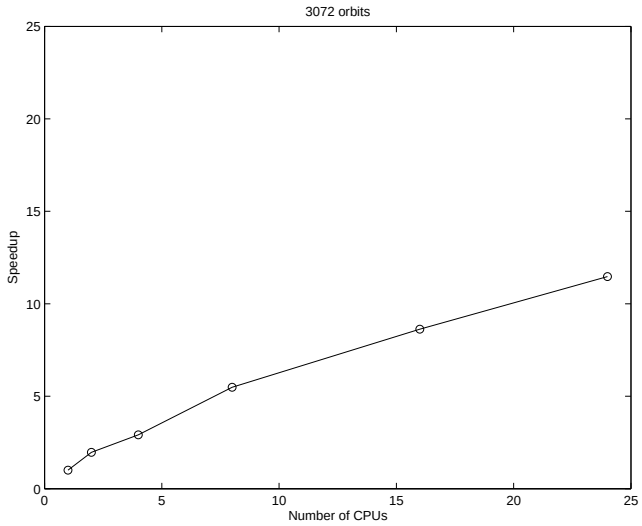


Figure 14: Parallelization relative speedup.

editors, *The Architecture of Scientific Software*, pages 145–158. Kluwer Academic Publishers, Boston, 2001.

[3] K. Åhlander and H. Munthe-Kaas. Applications of the Generalized Fourier Transform in Numerical Linear Algebra. *BIT Numerical Mathematics*, pages 819–850, 2005. Also available as Technical Report 2004-029, Department of Information Technology, Uppsala University.

[4] K. Åhlander and H. Munthe-Kaas. Eigenvalues for Equivariant Matrices. *Journal of Computational and Applied Mathematics*, 2005. In Press.

[5] E. L. Allgower, K. Böhmer, K. Georg, and R. Miranda. Exploiting symmetry in boundary element methods. *SIAM J. Numer. Anal.*, 29:534–552, 1992.

[6] E. L. Allgower, K. Georg, R. Miranda, and J. Tausch. Numerical exploitation of equivariance. *Zeitschrift für Angewandte Mathematik und Mechanik*, 78:185–201, 1998.

[7] Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenhaum, S. Hammarling, A. McKenney, D. Sorensen, and E. Anderson (editor). *LAPACK's user's guide*. SIAM, 3rd edition, 2000.

[8] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81:637–659, 1973.

[9] M. Bonnet. Exploiting partial or complete geometrical symmetry in 3D symmetric Galerkin indirect BEM formulations. *Intl. J. Numer. Meth. Engng*, 57:1053–1083, 2003.

[10] A. Bossavit. Boundary value problems with symmetry and their approximation by finite elements. *SIAM J. Appl. Math.*, 53:1352–1380, 1993.

[11] M. D. Buhmann. *Radial basis functions: theory and implementations*, volume 12 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2003.

[12] J. Cannon. A draft description of the group theory language Cayley. In *SYMSAC '76: Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 66–84, New York, NY, USA, 1976. ACM Press.

[13] S. Egner and M. Püschel. Symmetry-based matrix factorization. *Journal of Symbolic Computation*, 37(2):157–186, 2004.

[14] K. Georg and J. Tausch. A generalized Fourier transform for boundary element methods with symmetries. Technical report, Colorado State University, Ft. Collins, Colorado, 1994.

[15] K. Georg and J. Tausch. User's guide for a package to solve equivariant linear systems. Technical report, Colorado State University, Ft. Collins, Colorado, 1995.

[16] G. Golub and C. van Loan. *Matrix computations*. The John Hopkins Univeristy Press, 2nd edition, 1984.

[17] A. Hall. Pricing financial derivatives using radial basis functions and the generalized Fourier transform. Master's thesis, Uppsala University, Uppsala, Sweden, 2005.

[18] A. Hall, E. Larsson, and K. Åhlander. RBF-GFT option pricing (working title). Manuscript in preparation.

[19] S. Holmgren and K. Otto. A framework for polynomial preconditioners based on fast transforms I: Theory. *BIT Numerical Mathematics*, 38:544–559, 1998.

[20] S. Holmgren and K. Otto. A framework for polynomial preconditioners based on fast transforms II: PDE applications. *BIT Numerical Mathematics*, 38:721–736, 1998.

[21] M. Ljungberg and K. Åhlander. Generic Programming Aspects of Symmetry Exploiting Numerical Software. In P. Neittaanmäki et al., editors, *Proceedings of European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS 2004, Jyväskylä, 24–28 July 2004*, 2004. Also available as Technical Report 2004-020 from the Department of Information Technology, Uppsala University.

[22] J. S. Lomont. *Applications of Finite Groups*. Academic Press, New York, 1959.

[23] D. K. Maslen and D. N. Rockmore. Generalized FFTs - a survey of some recent results. In L. Finkelstein and W. Kantor, editors, *Proceedings of the 1995 DIMACS Workshop on Groups and Computation*, pages 183–237, June 1997.

[24] Object Management Group. *Unified Modeling Language: Superstructure, Formal Specification*, version 2.0, formal/05-07-04 edition, August 2005. `http://www.omg.org/cgi-bin/doc?formal/05-07-04`.

[25] D. N. Rockmore. Some applications of generalized FFTs. In L. Finkelstein and W. Kantor, editors, *Proceedings of the 1995 DIMACS Workshop on Groups and Computation*, pages 329–369, June 1997.

[26] D. N. Rockmore. Recent progress and applications in group FFTs. In *Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 773–777. NATO Advanced Study Institute on Computational Noncommutative Algebra and Applications, IEEE, November 2002. ISSN: 1058-6393.

[27] J. Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[28] A. Seress. An introduction to computational group theory. *Notices of the AMS*, 44(6):671–679, June/July 1997.

[29] J. P. Serre. *Linear Representations of Finite Groups*. Springer-Verlag, 1977.

[30] L. Stiller. *Exploiting symmetry on parallel architectures*. PhD thesis, Johns Hopkins University, 1995.

[31] Sun Microsystems. *UltraSPARC III Cu User's Manual*, 2.2.1 edition, January 2004. `http://www.sun.com/processors/UltraSPARC-III/index.xml`.

[32] J. Tausch. Equivariant preconditioners for boundary element methods. *SIAM Sci. Comp.*, 17:90–99, 1996.

[33] A. Trønnes. Symmetries and generalized Fourier transforms applied to computing the matrix exponential. Master's thesis, University of Bergen, Bergen, Norway, 2005.

[34] A. Yamba Yamba. The generalized Fourier transform applied to equivariant systems. Master's thesis, Uppsala University, Uppsala, Sweden, 2005.

A The Fast algorithms

The exact implementations of the **Fast 1** and **Fast 2** algorithms are given in Algorithms 6 and 7, respectively. The elements of $\mathcal{D}_4$ are enumerated in the following order: $e, a, a^2, a^3, b, ab, a^2b, a^3b$. Thus, $u(0)$ should be interpreted as $u(e)$, $u(1)$ as $u(a)$, etc. In Fourier space, $\hat{u}(0)$ refers to $\hat{u}(\mathcal{R}_0)$, etc.

Algorithm 6 Fast 1 computation of $\widehat{u} = \mathrm{fgft}_1(u)$

$t_1 = u(0) + u(2)$
$t_2 = u(1) + u(3)$
$t_3 = u(4) + u(6)$
$t_4 = u(5) + u(7)$
$t_5 = u(0) - u(2)$
$t_6 = u(1) - u(3)$
$t_7 = u(4) - u(6)$
$t_8 = u(5) - u(7)$
$t_9 = t_1 + t_2$
$t_{10} = t_1 - t_2$
$t_{11} = t_3 + t_4$
$t_{12} = t_3 - t_4$
$\widehat{u}(0)_{0,0} = t_9 + t_{11}$
$\widehat{u}(1)_{0,0} = t_9 - t_{11}$
$\widehat{u}(2)_{0,0} = t_{10} + t_{12}$
$\widehat{u}(3)_{0,0} = t_{10} - t_{12}$
$\widehat{u}(4)_{0,0} = t_5 - t_7$
$\widehat{u}(4)_{1,0} = -t_6 - t_8$
$\widehat{u}(4)_{0,1} = t_6 - t_8$
$\widehat{u}(4)_{1,1} = t_5 + t_7$

Algorithm 7 Fast 2 computation of $\widehat{u} = \mathrm{fgft}_2(u)$

$\widehat{u}(3)_{0,0} = u(0) + u(2)$
$t_1 = u(4) + u(6)$
$t_2 = u(1) + u(3)$
$t_3 = u(5) - u(7)$
$\widehat{u}(1)_{0,0} = \widehat{u}(3)_{0,0} + t_2$
$\widehat{u}(3)_{0,0} = \widehat{u}(3)_{0,0} - t_2$
$t_4 = t_1 + t_3$
$t_1 = t_1 - t_3$
$\widehat{u}(0)_{0,0} = \widehat{u}(1)_{0,0} + t_4$
$\widehat{u}(2)_{0,0} = \widehat{u}(3)_{0,0} + t_1$
$\widehat{u}(3)_{0,0} = t_1$
$\widehat{u}(4)_{1,1} = u(0) - u(2)$
$t_5 = u(1) - u(3)$
$t_6 = u(4) - u(6)$
$\widehat{u}(4)_{0,1} = u(7) - u(5)$
$\widehat{u}(0)_{0,0} = \widehat{u}(4)_{1,1} - t_6$
$\widehat{u}(4)_{1,1} = \widehat{u}(4)_{1,1} + t_6$
$\widehat{u}(4)_{1,0} = \widehat{u}(4)_{0,1} - t_5$
$\widehat{u}(4)_{0,1} = t_5$