Genetic Programming and Evolvable Machines (2022) 23:253–278
https://doi.org/10.1007/s10710-021-09425-5

Genetic programming for iterative numerical methods

Dominik Sobania[1] · **Jonas Schmitt**[2] · **Harald Köstler**[2] · **Franz Rothlauf**[1]

Received: 3 December 2020 / Revised: 15 September 2021 / Accepted: 25 September 2021 /
Published online: 25 November 2021

Abstract
We introduce GPLS (Genetic Programming for Linear Systems) as a GP system that finds mathematical expressions defining an iteration matrix. Stationary iterative methods use this iteration matrix to solve a system of linear equations numerically. GPLS aims at finding iteration matrices with a low spectral radius and a high sparsity, since these properties ensure a fast error reduction of the numerical solution method and enable the efficient implementation of the methods on parallel computer architectures. We study GPLS for various types of system matrices and find that it easily outperforms classical approaches like the Gauss–Seidel and Jacobi methods. GPLS not only finds iteration matrices for linear systems with a much lower spectral radius, but also iteration matrices for problems where classical approaches fail. Additionally, solutions found by GPLS for small problem instances show also good performance for larger instances of the same problem.

Keywords Genetic programming · Iterative numerical methods · Linear systems · Sparse linear algebra

✉ Dominik Sobania
dsobania@uni-mainz.de

Jonas Schmitt
jonas.schmitt@fau.de

Harald Köstler
harald.koestler@fau.de

Franz Rothlauf
rothlauf@uni-mainz.de

[1] Johannes Gutenberg University Mainz, Mainz, Germany

[2] Friedrich–Alexander University Erlangen–Nürnberg, Erlangen, Germany

1 Introduction

Numerical methods are used in various disciplines to solve problems where an analytical solution does not exist or is difficult to find. In computational science and engineering, for example, one tries to model physical phenomena and then to approximate these usually continuous mathematical models numerically. The computation of a numerical solution often requires solving a system of (non-)linear equations. Since the number of unknowns can be huge in numerous real-world applications, efficient and scalable solvers for such systems are necessary. Unfortunately, the optimal solver method depends on the system of equations itself and therefore it is impossible to formulate a single algorithm for this purpose. However, over the past decades, several numerical solvers have been proposed in the field of applied mathematics, which are usually efficient for certain classes of system matrices (the coefficient matrix of a linear system) [3, 21].

Genetic programming (GP) is an evolutionary computation technique that has been successfully applied to various real-world problems during the last decades [18]. Especially in the field of symbolic regression, where the aim is to find mathematical expressions solving a given problem, GP has been used to approximate even complex problems [12, 27]. This makes GP an interesting approach for finding new iterative numerical methods as it can be used to find the required mathematical expressions to generate iteration matrices based on certain classes of given system matrices.

This paper applies a novel GP approach for linear systems (GPLS); an approach that finds an iteration matrix for a given linear system. To ensure that the resulting iterative numerical method can be executed efficiently on parallel computer architectures, we are interested in a low spectral radius and a high sparsity of the found iteration matrices. We evaluate the found methods on standard test problems and real-world use cases to demonstrate the human competitiveness of GPLS and to compare it with traditional methods.

GPLS uses certain elements (e.g., some functions and terminals) previously proposed in a short paper by Mahmoodabadi et al. [15] in their presentation of a first prototype for solving linear systems with GP, and by Schmitt et al. [22] who focus on special classes of sparse linear systems. The aim of GPLS is to find a good iteration matrix based on an input system matrix in general and not one that only serves special cases. The iteration matrix is the core component of all considered numerical methods. For GPLS, we define an objective function that measures the generated iteration matrices' spectral radius and the sparsity, as well as the method's complexity (size of the generated mathematical term equal to the number of tree nodes). The spectral radius is an indicator for the convergence of the generated method, high sparsity provides performance advantages in the calculation and implementation of the method, and the complexity measure serves as bloat control.

Following this introduction, we present a background to iterative numerical methods and explain the relevant stationary iterative numerical methods in Sect. 2, describe the discretization of partial differential equations to systems of

linear equations in Sect. 3, introduce GPLS in detail in Sect. 4, and present our experiments and discuss the results in Sect. 5. Section 6 concludes the paper.

2 Iterative numerical methods

The most fundamental problem within linear algebra is finding the solution of the linear system $Ax = b$, where $A \in \mathbb{R}^{m \times n}$ is the *coefficient matrix*, $x \in \mathbb{R}^m$ the *vector of unknowns*, and $b \in \mathbb{R}^m$ the *right-hand side vector*. If A is a squared nonsingular (or invertible) matrix, there exists a single unique solution x^* of the system.

Most linear systems derived from science and engineering phenomena do possess certain special structures. The most eminent property of these systems is *sparsity* which means that the majority of the entries of the coefficient matrix A are zero while the number of nonzero entries is usually of order n. Sparsity significantly reduces the number of required elementary matrix operations when solving the linear problem. For instance, assuming that matrix A has αn nonzero entries, a matrix-vector multiplication can be performed in $\mathcal{O}(\alpha n)$ operations. Therefore, the design of efficient algorithms for solving these systems relies heavily on exploiting the sparsity of the coefficient matrix A.

In general, methods for solving linear systems can be classified either as *direct* or as *iterative* methods. Direct methods require only a finite number of steps. An example is Gaussian elimination, the standard textbook method for solving an arbitrary linear system of equations. The commonality of all direct methods is that they directly manipulate the individual entries of A and thus need to operate on an explicit representation of the matrix. Transformations applied within direct solvers such as Gaussian or Householder triangulation do not preserve the sparse structure of A [25].

In contrast, iterative methods perform successive approximations to a linear system to obtain more accurate solutions. Typically, these approximations only require the calculation of matrix-vector products, where all matrices involved can be derived from the system matrix without destroying its sparsity. As a result, although specialized direct methods for solving sparse systems exist [5], the largest, currently considered systems are solved using iterative methods [2, 25].

The two main classes of iterative methods for solving systems of linear equations are *stationary* and *non-stationary* methods whereby most of the latter belong to the subclass of *Krylov subspace* methods. Stationary methods solve a linear system by repeatedly applying an iteration matrix, derived from the coefficient matrix A, to an initial guess for the vector of unknowns x, to get a series of approximations that converge to the actual solution of the system. Stationary methods have the advantage that they are easier to implement and to analyze than non-stationary methods. This paper focuses exclusively on stationary methods to automatically generate iterative solvers for sparse linear systems.

In the following, we provide a brief overview of the stationary iterative methods. For a more comprehensive overview of the iterative methods, including non-stationary methods, see [3, 7, 21], and [6].

Stationary iterative methods are expressed in the general form

$$x^{(k+1)} = Gx^{(k)} + f, \tag{2.1}$$

where G is the *iteration matrix*, $x^{(k)}$ the solution vector in iteration k (i.e. the current iterate), and f a vector that is obtained by transforming the right-hand side b. Neither G nor f depends on the iteration count. The standard iterative methods are Jacobi and Gauss–Seidel.

2.1 The Jacobi method

To derive the Jacobi method we consider the equations of the linear system $Ax = b$ in isolation, which leads to

$$\sum_{j=1}^{n} a_{ij}x_j = b_i.$$

By solving each of the equations for x_i we obtain

$$x_i = \left(b_i - \sum_{j \neq i} a_{ij}x_j \right) / a_{ii}.$$

If we assume that all entries except x_i are fixed in every individual equation, the iterative scheme is defined by

$$x_i^{(k+1)} = \left(b_i - \sum_{j \neq i} a_{ij}x_j^{(k)} \right) / a_{ii}.$$

This iteration is the scalar formulation of the Jacobi method in which $x_i^{(k)}$ corresponds to the ith component of the solution vector in iteration k and a_{ij} to the entry of A in row i and column j. Since all equations are treated independently, the order of examination is irrelevant. In fact, $x_i^{(k+1)}$ can be computed simultaneously for all equations, which makes the Jacobi method easily parallelizable.

To provide a definition of the Jacobi method in matrix form, we first introduce the splitting

$$A = D - L - U,$$

where D is the diagonal, L the strictly lower triangular and U the strictly upper triangular part of A. The term *strictly* refers to the fact that the diagonal of A is excluded. We assume that all diagonal entries of A are nonzero. Using this splitting, the matrix form of the Jacobi method is obtained by

$$x^{(k+1)} = D^{-1}(L + U)x^{(k)} + D^{-1}b.$$

Note that the inverse of a diagonal matrix is a diagonal matrix with the original diagonal entries inverted. The iteration, in addition, corresponds to our basic formulation of stationary iterative methods (see Eq. 2.1) with the iteration matrix

$$G = D^{-1}(L + U) = I - D^{-1}A,$$

and

$$f = D^{-1}b.$$

2.2 The Gauss–Seidel method

The Jacobi method can simultaneously compute the new iterate for all components of the solution vector. In contrast, the Gauss–Seidel method examines them in sequence, such that already computed components are taken into account:

$$x_i^{(k+1)} = \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right) / a_{ii}.$$

The computation of each new component depends on the previously computed components and cannot be performed simultaneously. While this implies a serialization of the computation, the order can be varied. Different orders will inevitably lead to different values of the new iterate $x^{(k+1)}$ affecting the overall convergence of the method. Therefore, the Gauss–Seidel method's serial nature in general prohibits a parallel computation. However, when the matrix A is sparse, not all components of the new iterate $x^{(k+1)}$ depend on the values of all components of the old iterate. Then, it is possible to define a partitioning of x such that there are no dependencies between the components in the same partition and consequently the Gauss–Seidel method can be applied to each partition in parallel. For a more detailed discussion of the parallelization of the Gauss–Seidel method, see [21]. The matrix formulation of the Gauss–Seidel method is defined by

$$x^{(k+1)} = (D - L)^{-1} U x^{(k)} + (D - L)^{-1} b.$$

Note that the iteration contains the computation of the inverse of $(D - L)$, which is a lower triangular matrix. A multiplication with the inverse of this matrix corresponds to solving a linear system via backward substitution and therefore does not require the explicit computation of the inverse. The Gauss–Seidel method can also be formulated as a stationary iterative method (see Eq. 2.1) with the iteration matrix

$$G = (D - L)^{-1} U = I - (D - L)^{-1} A,$$

and

$$f = (D - L)^{-1} b.$$

2.3 Successive over-relaxation

The successive over-relaxation (SOR) method [29] extends the Gauss–Seidel method by applying a weighted average between the previous and the newly computed iterate. It is defined by

$$x_i^{(k+1)} = \omega x_i^{(k+1)} + (1-\omega)x_i^{(k)}.$$

The idea is to choose ω in a way that accelerates the convergence of the method to the actual solution. Note that if $\omega = 1$, SOR corresponds to the Gauss–Seidel method. SOR only converges for values of $\omega \in (0, 2)$ [11]. In general it is not possible to estimate the optimal value of ω a priori and therefore a heuristic is usually employed to choose an ω. Like the Jacobi and Gauss–Seidel methods, the SOR method can also be defined in terms of matrices and vectors by the iteration

$$x^{(k+1)} = (D - \omega L)^{-1}(\omega U + (1-\omega)D)x^{(k)} + \omega(D - \omega L)^{-1}b.$$

Similar to Gauss–Seidel, all inverted matrices are lower triangular matrices and the respective matrix vector products can be obtained via backward substitution without explicitly computing the inverse. For the complete derivation of SOR, see [29].

2.4 Convergence of stationary methods

Both, the Jacobi method and the Gauss–Seidel method define a sequence of approximations of the basic form as defined in Eq. 2.1. In case convergence is reached, the limit x of this iteration satisfies

$$x = Gx + f. \tag{2.2}$$

Essential for the convergence of stationary iterative methods is the spectral radius ρ of the iteration matrix G defined by

$$\rho(G) = \max_{1 \le j \le n} |\lambda_j(G)|, \tag{2.3}$$

where $\lambda_j(G)$ are the eigenvalues of G.

Theorem 1 *Let G be a square matrix with spectral radius $\rho(G)$, then*

$$\lim_{k \to \infty} G^k = 0$$

if and only if $\rho(G) < 1$.

For a proof, see [21]. Equation 2.2 can be reformulated into the following system of linear equations:

$$x - Gx = f, \tag{2.4}$$

$$(I - G)x = f. \tag{2.5}$$

Equation 2.5 has a unique solution x^* if and only if $(I - G)$ is a non-singular square matrix. Subtracting Eq. 2.2 from the basic iteration scheme presented in Eq. 2.1 leads to

$$x^{(k+1)} - x^* = G(x^{(k)} - x^*) = \ldots = G^{k+1}(x_0 - x^*).$$

From Theorem 1 it follows that the sequence $x^{(k+1)} - x^* = G^{k+1}(x_0 - x^*)$ converges to zero. Therefore, we can conclude about the convergence of an arbitrary stationary iterative method:

Theorem 2 *Let G be a square matrix with $\rho(G) < 1$, then $I - G$ is non-singular and the iteration presented in Eq. 2.1 converges for any f and x_0. Conversely, if the iteration presented in Eq. 2.1 converges for any f and x_0, then $\rho(G) < 1$.*

As Theorem 2 states, the convergence of any stationary iterative method solely depends on finding an iteration matrix with a spectral radius smaller than one. Furthermore, the general *convergence factor* of an iterative method is equal to the spectral radius of its iteration matrix [21]. Therefore, the design of an efficient iterative method is equivalent to finding an iteration matrix with a low (or even minimal) spectral radius.

Since the computation of the spectral radius is expensive, it is often not practical to compute it directly. For Jacobi, Gauss–Seidel and other well-studied methods there exist certain criteria for the system matrix A that ensure the convergence of these methods [7]. One possibility to estimate the spectral radius for arbitrary stationary iterative methods is the use of Local Fourier Analysis (LFA) [20, 28].

3 Discretization of partial differential equations

Many problems in science and engineering can be mathematically modeled in the form of a *partial differential equation (PDE)*. A classic example is the Navier–Stokes equation that describes the motion of a viscous fluid [24] and can be used to model a wide range of phenomena, with applications ranging from weather forecasting to aircraft design. Although there exists a rich theory about PDEs, only a few cases of analytical solutions for these equations are known. As a remedy, numerical methods can be applied to approximate the solution of a PDE at a finite number of points what transforms the problem of solving a PDE into a problem of solving a system of linear equations. This transformation is usually referred to as *discretization*. The most widely used methods of discretizing a PDE are the *finite difference method (FDM)*, the *finite volume method (FVM)*, and the *finite element method (FEM)*. To provide a brief introduction to the discretization of PDEs, we focus on FDM. For more information on the numerical solutions of PDEs, see [1, 10, 17, 23], and [26].

One of the most basic but also quite common PDE is Poisson's equation which is defined as

$$\Delta u = f,$$

where Δ is the Laplace operator, u and f are real or complex-valued functions. Typically, f is given and one wants to solve the equation for u. For $u, f \in \mathbb{R}^2$ it takes the form

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u(x, y) = f(x, y). \tag{3.1}$$

If u is differentiable at a point x, we can create the Taylor expansion

$$u(x + h, y) = u(x, y) + h u_x(x, y) + \frac{h^2}{2} u_{xx}(x, y) + \frac{h^3}{6} u_{xxx}(x, y) + \mathcal{O}(h^4) \tag{3.2}$$

$$u(x, y + h) = u(x, y) + h u_y(x, y) + \frac{h^2}{2} u_{yy}(x, y) + \frac{h^3}{6} u_{yyy}(x, y) + \mathcal{O}(h^4), \tag{3.3}$$

in which u_x and u_y denote the first order partial derivatives of u with respect to x and y, respectively.

$$u_x(x, y) = \frac{\partial}{\partial x} u(x, y), \; u_{xx}(x, y) = \frac{\partial^2}{\partial x^2} u(x, y), \ldots$$

When stopping the Taylor expansion after the third term, the error that results from this approximation is of order $\mathcal{O}(h^4)$. Similarly, we can define

$$u(x - h, y) = u(x, y) - h u_x(x, y) + \frac{h^2}{2} u_{xx}(x, y) - \frac{h^3}{6} u_{xxx}(x, y) + \mathcal{O}(h^4) \tag{3.4}$$

$$u(x, y - h) = u(x, y) - h u_y(x, y) + \frac{h^2}{2} u_{yy}(x, y) - \frac{h^3}{6} u_{yyy}(x, y) + \mathcal{O}(h^4). \tag{3.5}$$

Adding Eq. 3.2 to Eq. 3.4 and Eq. 3.3 to Eq. 3.5 and dividing by h^2 yield the following approximation for the second order partial derivative of u:

$$\frac{\partial^2}{\partial x^2} u(x, y) = \frac{u(x + h, y) + u(x - h, y) - 2u(x, y)}{h^2} + \mathcal{O}(h^2) \tag{3.6}$$

$$\frac{\partial^2}{\partial y^2} u(x, y) = \frac{u(x, y + h) + u(x, y - h) - 2u(x, y)}{h^2} + \mathcal{O}(h^2) \tag{3.7}$$

In both cases, the approximation error is of order $\mathcal{O}(h^2)$. Although not covered here, the approximation of first or higher-order derivatives can be defined in a similar fashion. Equations 3.6 and 3.7 can now be used to define an approximation for the Laplace operator. This results in the discrete version of Poisson's equation (see Eq. 3.1)

$$\frac{1}{h^2}(u(x+h,y)+u(x-h,y)+u(x,y+h)+u(x,y-h)-4u(x,y))=f.$$

Consequently, to compute the solution of Poisson's equation on an arbitrary two dimensional domain using a finite difference approximation, a system of linear equations must be solved, whereby the solution at each point is represented by an equation of the form of the discrete version of Poisson's equation. A common decision is to choose a uniform h for the whole domain, such that the individual equations are independent of its value.

In order to solve the system, an additional set of equations must be defined that defines how the system behaves at the boundaries of the domain. These *boundary conditions* usually come in three types:

- Dirichlet condition: $u(x)=\phi(x)$
- Neumann condition: $\frac{\partial}{\partial n}u(x)=0$
- Cauchy condition: $\frac{\partial}{\partial n}+\alpha(x)u(x)=\gamma(x)$

The vector n refers to a unit vector normal to the domain and directed outwards. In many cases, boundary conditions are of mixed type, which means that at different parts of the boundary different conditions are defined. As we do not explicitly treat boundary conditions in this work, we assume that the boundary conditions are contained in the right-hand side vector b. Thus, our derivation results in a linear system of the form

$$Ax=b.$$

We present an example and assume a 5x5 grid with 9 interior grid points, Dirichlet conditions at all boundaries, and a natural ordering of the grid points. This problem can, for instance, result in a linear system with the matrix A and right-hand side b:

$$A=\begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0\\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0\\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0\\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0\\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0\\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1\\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0\\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1\\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix},\quad b=\begin{bmatrix} -h^2f_{22}+u_{12}+u_{21}\\ -h^2f_{32}+u_{31}\\ -h^2f_{42}+u_{52}+u_{41}\\ -h^2f_{23}+u_{13}\\ -h^2f_{33}\\ -h^2f_{43}+u_{53}\\ -h^2f_{24}+u_{14}+u_{25}\\ -h^2f_{34}+u_{35}\\ -h^2f_{44}+u_{54}+u_{45} \end{bmatrix}$$

f_{ij} and u_{ij} denote the values of u and f at position (ih, jh) within the domain. Note that A is a sparse band matrix. Therefore, we expect an iterative solver to preserve this property when computing an approximate solution for the system. Approximating Poisson's equation with finite differences in one or three dimensions can be performed in a similar manner, which also results in linear systems of the form $Ax=b$ with a band matrix A.

4 GPLS: genetic programming for linear systems

GPLS is a standard GP approach which aims to find mathematical expressions that define an iteration matrix. It uses the system matrix A of a linear system as input. In contrast to classical regression approaches [13, 14], the solutions of GPLS are iteration matrices G and we have no training points given. Our aim is to find iteration matrices G with a low spectral radius $\rho(G)$. When obtaining such an iteration matrix G, we can use it with iterative methods to find an approximate solution for a linear system.

4.1 Representation of iteration matrices

We use a tree-based representation to describe an iteration matrix by a mathematical term. The result of this term is the iteration matrix. We use a function and terminal set that allow the application of well-known iterative numerical methods [3]. Accordingly, the function set is defined as

$$\{+, -, *\}.$$

The terminal set contains multiple variations of the system matrix A, which can be calculated offline before the start of a GP run. For our experiments, we use

$$\{A, D, D^{-1}, (A - D), (L + D), (L + D)^{-1}, U\},$$

where A is the system matrix, D is the diagonal matrix of A, L is the strictly lower triangular part of A, and U is the strictly upper triangular part of A.

We do not use the right hand side vector b in the terminal set because it is not necessary for the convergence of the used iterative method as defined in Theorem 2 [21]. Furthermore, the omission of the vector b allows us to work exclusively on square matrices, so it is not required to use a strongly typed GP (STGP) [16] with custom addition and subtraction operations for matrices and vectors.

Figure 1 shows an example tree representation for a candidate iteration matrix described by the term $(A - D) + D^{-1}(D^{-1} + U)$. The leaves contain the terminals which are the system matrix A and pre-calculated variations of A (the tree's lowest level in Fig. 1 depicts this for an example system matrix).

4.2 Objective function

The spectral radius of the iteration matrix G determines the convergence behavior of stationary methods (Eq. 2.3). Iterative methods converge if $0 < \rho(G) < 1$ holds and G is not a diagonal matrix [21], which makes the spectral radius the most important part of the objective function. In addition, we want to increase the sparsity of the iteration matrix. Therefore, our objective function rewards a high number of zeros in the iteration matrix. Finally, we also want to keep the complexity

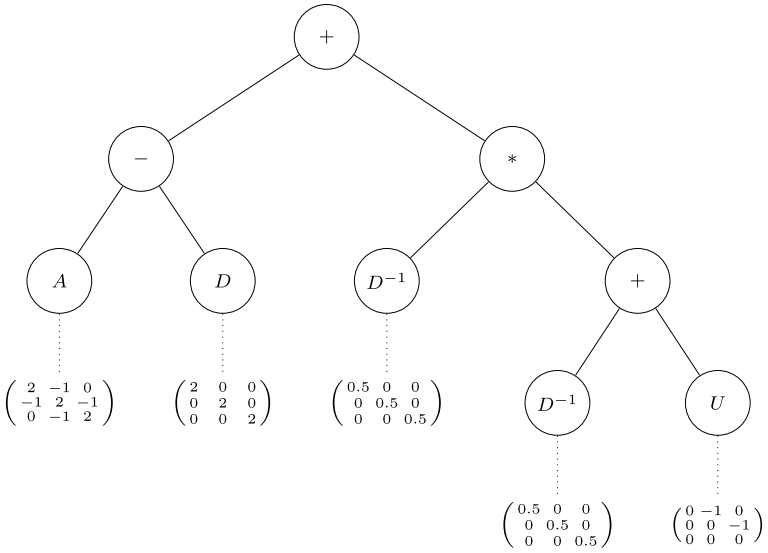


Fig. 1 An example tree representation for a candidate iteration matrix described by the term $(A - D) + D^{-1}(D^{-1} + U)$

of the mathematical term under control. We combine these three goals into a single objective function and obtain

$$f(s, c, z) = \begin{cases} w_s s/s_{max} + w_z z/z_{max} + w_c c/c_{max} & \text{if } \rho(G) > 0 \wedge G \notin diag(\mathbb{R}^{N \times N}) \\ w_s + w_z z + w_c c & \text{else} \end{cases}, \tag{4.1}$$

where s is the spectral radius ρ of the candidate iteration matrix, z is the number of non-zero entries in the considered candidate iteration matrix, and c is the number of nodes in the tree representing the candidate iteration matrix. s_{max}, z_{max} and c_{max} are the largest observed values. $w_c c/c_{max}$ measures the number of nodes (complexity) of the expression's parse tree and serves as bloat control. This kind of bloat control is a variant of the well-known parsimony pressure [4, 19]. We assume a minimization problem. The coefficients w_s, w_z, and w_c are real-valued weights in the interval [0, 1] such that $w_s + w_z + w_c = 1$. As mentioned before, for the convergence of the iterative methods, we must, in addition to a low spectral radius, ensure that $\rho(G) \neq 0$ and G is not a diagonal matrix. Individuals that violate these conditions are penalized by setting $s/s_{max} = 1$.

As the calculation of the spectral radius ρ is the component that determines the computational effort of the fitness function and consequently also of an entire GP run, we measure the run-time in milliseconds required for the calculation of the spectral radius ρ for random square matrices for increasing problem sizes n. Figure 2 shows box-plots of the time in milliseconds required to calculate the spectral radius ρ over problem size n. We use Python's Numpy module to calculate the spectral radius ρ and measure the run-times using an AMD Ryzen Threadripper 3990X (4.3 GHz maximum boost clock) with 64 cores. For every problem size n, we report 100

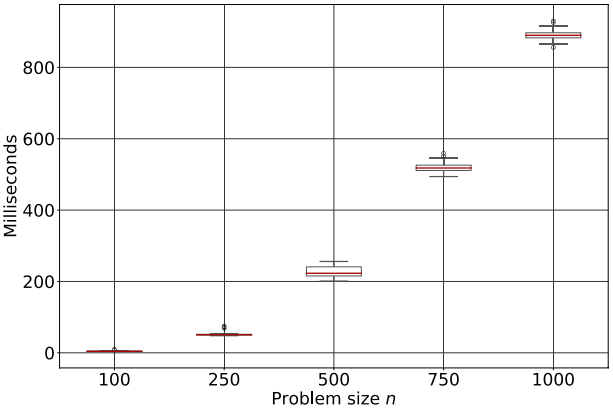


Fig. 2 Box-plots of the time in milliseconds required to calculate the spectral radius ρ over problem size n

time measurements. The plots show that the required run-time to calculate the spectral radius ρ increases notably with the problem size n. However, even for matrices of size 1000×1000, the median run-time is still lower than one second.

5 Experiments and results

We study the performance of GPLS and the resulting iterative methods on various standard test problems with randomly generated dense system matrices, different types of randomly generated band matrices, and real-world applications.

For comparability, we use the same settings for our GP approach in all experiments. The individuals in the first generation ($i = 0$) are generated by the ramped-half-and-half method. As variation operators, we use standard subtree-crossover with a crossover probability of $p_c = 0.8$ and standard subtree-mutation with a mutation probability of $p_m = 0.05$. For selection, we use tournament selection of size 3. The population size is set to 1, 500 and we stop a GP run after 30 generations.

The weights for the objective function were determined based on some manual parameter tuning. We set the weight for the spectral radius to $w_s = 0.8997$, the weight for the non-zero values to $w_z = 0.1$, and method's complexity weight (nodes in the parse tree) to $w_c = 0.0003$. The largest value is assigned to w_s because we require $\rho(G) < 1$ to guarantee convergence of the iterative method. To favor small solutions, we assign a very low value to w_c.

5.1 Performance of GPLS for random system matrices

The main application of the iterative methods found by GPLS is the solution of a linear system discretized from PDEs. However, for a first analysis of the GP performance we use randomly generated system matrices as input. We study the change of the three components of the objective function—spectral radius,

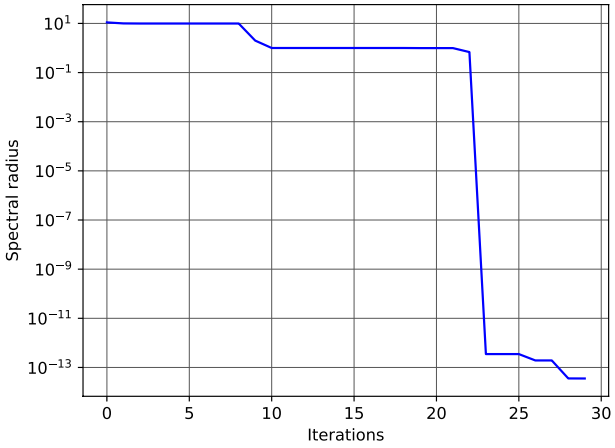
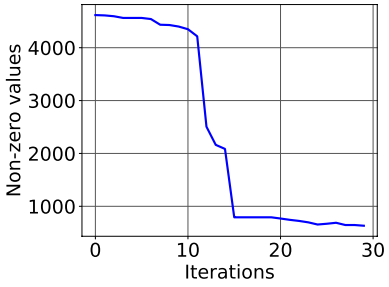


Fig. 3 Spectral radius over iterations for a 100×100 matrix

Fig. 4 Non-zero values over iterations for a 100×100 matrix



non-zero values, and the number of nodes in the parse tree—for the best individual over time in a GP run on randomly generated system matrices of increasing size. To generate a random system matrix, for a given matrix size we fill the elements with equally distributed integer values ranging from -10 to 10.

Figure 3 shows the median spectral radius $\tilde{\rho}$ of 100 GP runs over the number of generations. As input we use a randomly generated 100×100 dense system matrix. We find that the average median of the spectral radius decreases from about 11 (at the beginning of the runs) to about 3e$-$14 at the end of the runs.

For the same randomly generated 100×100 system matrix used as input, Figs. 4 and 5 show the median non-zero entries $\tilde{z}$ and median number of nodes $\tilde{c}$, respectively. We find that the number of non-zero entries decrease over the run with a strong reduction between generations 10 and 15. The median number of nodes increases slightly over a run starting with about seven tree nodes and increasing to 13 nodes. Due to the low weight w_c for parsimony pressure, the number of nodes slightly increases while still allowing GPLS to improve on the spectral radius and sparsity. This is reflected by the choice of the weights in the objective function, where spectral radius and sparsity are more important than the size of the resulting iterative numerical method ($w_s, w_z > w_c$).

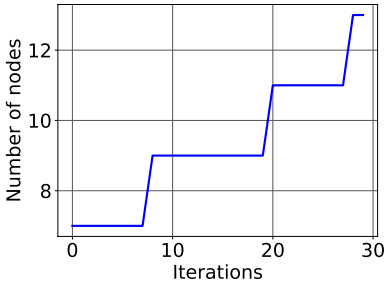


Fig. 5 Complexity over iterations for a 100×100 matrix

Table 1 extends the analysis and presents results for the spectral radius, the number of non-zero entries, and the number of nodes in the parse tree for random problems of size 10×10 to 100×100. For each problem size, we perform 100 runs with a random system matrix. We show the median as well as the interquartile range (IQR; in parentheses) of the best solution in the initial ($i = 0$) and last generation ($i = 29$). We use the IQR as a proxy for the variance of the results. It is defined as the difference between the 75th and the 25th percentile. Best median results of a

Table 1 Median and interquartile range of spectral radius, number of non-zero entries, and number of nodes in the parse tree (method's complexity) in first ($i = 0$) and last ($i = 29$) generation. We present results for different problem sizes

Problem size	Spectral radius $\tilde{s}$		Non-zero entries $\tilde{z}$		Number of nodes $\tilde{c}$	
	$i = 0$	$i = 29$	$i = 0$	$i = 29$	$i = 0$	$i = 29$
10×10	9.0 (0.111)	**0.333** (0.333)	53 (7)	**46** (42)	**1** (6)	7 (12)
20×20	9.95 (89.099)	**1.776e−15** (0.999)	191 (12)	**65** (119.25)	**7** (4)	15 (14)
30×30	9.945 (9.0)	**1.776e−15** (0.999)	432 (14)	**168** (290.25)	**7** (2)	15 (10)
40×40	10 (89.099)	**8.881e−16** (0.528)	692.5 (93)	**120** (389.75)	**7** (2)	17 (12)
50×50	10 (89.347)	**0.75** (0.999)	1147.5 (73)	**194** (1013.75)	**7** (2)	15 (12)
60×60	10.95 (89.099)	**1.191e−15** (0.999)	1657 (133)	**235** (991)	**7** (6.5)	15 (8.5)
70×70	10 (100)	**0.187** (9.899)	2191.5 (225)	**1076** (1885)	**7** (4)	15 (10)
80×80	10 (90.092)	**0.328** (0.992)	2962 (276)	**735** (2386.5)	**7** (2)	13 (8.5)
90×90	10.9 (90.092)	**1.139e−14** (0.999)	3745 (168)	**538** (1772.25)	**7** (3)	15 (8)
100×100	11 (100)	**3.554e−14** (0.999)	4617 (148.25)	**632.5** (4005)	**7** (2)	13 (10)

run are printed in bold. All differences between the first and last generations were tested for significance with a Wilcoxon rank-sum test ($p < 0.001$).

We find that GPLS reliably finds solutions with low spectral radius (median spectral radius $\tilde{\rho} < 1.0$ for all studied problem instances). For some problem sizes, we observe a quite large IQR because the search space is complex and GPLS does not always find a successful solution (where $\rho < 1.0$). However, this is not a problem for the practical use of GPLS, since we can simply check the found solution for its suitability (calculate the spectral radius) and, if necessary, restart the GPLS run. In addition to the spectral radius, the GP approach also improves the sparsity of the found iteration matrices for all problem sizes. Only the number of nodes increase during a GP run. This is expected as the weight w_c is chosen very low to work only as slight bloat control, as a median size of 15 nodes is acceptable (comparable to the Gauss–Seidel and the Jacobi methods).

5.2 Generalization of iteration matrices found by GPLS

A direct comparison of GPLS and classical stationary iterative methods is difficult as GPLS' main effort comes from the search for a suitable term that builds an iteration matrix from a system matrix. This effort is high, especially if the considered linear systems are very large. In contrast, classical stationary iterative methods like Gauss–Seidel do not require any search process but are directly applicable.

A relevant question is whether GPLS finds iteration matrices that are general and can (analogously to classical stationary iterative methods) be applied to a wide range of different problems. When searching for such generalizable expressions, we can utilize the fact that linear systems discretized from PDEs often have similar structures and characteristics independently of their degree of detail and size. We can take advantage of this and evolve iteration matrices with GPLS for small linear systems and subsequently use the found solutions on larger systems with a similar structure, based on the assumption that the found solutions for the small systems also yield satisfactory results for the larger systems.

We study the generalization of the found solutions with a set of diagonal $n \times n$ band matrices used as system matrices, which are also relevant for real-world problems (see tridiagonal Toeplitz matrices [8]). A band matrix is a sparse matrix with a main diagonal and additional diagonals on both sides of the main diagonal containing non-zero values [7]. We use diagonal matrices in 1D and 2D with additional diagonals on the upper side and on the lower side of the main diagonal [9]. The structure of these matrices is independent of the node size n because, for each matrix, we use consistent values for the diagonals.

In our experiments, we randomly generate 100 system matrices of low size ($n = 5$ and $n = 9$). For each of the problems, GPLS determines an iteration matrix. In a next step, for each of the 100 system matrices (for each considered problem type) we generate appropriate system matrices with larger n. The larger system matrices are also diagonal matrices. We apply the solution that has been found by GPLS for the low value of n to the larger system matrices and evaluate the corresponding spectral

radii ρ of the iteration matrices. Our hope is that the solutions found for small n are general and also work well for larger n.

Figure 6 shows box-plots of the spectral radius ρ over the problem size n of the $n \times n$ system matrices. Each box-plot contains the spectral radius of 100 iteration matrices. The dashed line shows a spectral radius of 1.0. In this experiment, GPLS was only applied to diagonally dominant and diagonal system matrices in 1D of size 5×5. Thus, only the spectral radii of the iteration matrices in the first box-plot are a direct result of GPLS. And for this first box-plot, we considered only found iteration matrices with a spectral radius $\rho < 1.0$. For the larger system matrices, we did not apply GPLS anew but re-used the iterative methods evolved for the small system matrices ($n = 5$).

As expected, the spectral radii become larger with increasing n. Nevertheless, the median spectral radius is always lower than 1.0 for the analyzed matrix sizes. For $n = 5$, GPLS finds solutions with a median spectral radius $\rho = 9.23e - 6$. Applying these solutions to a problem with $n = 1000$ still yields a median spectral radius $\tilde{\rho} < 1.0$.

Figure 7 shows the same analysis, but this time we start from 9×9 diagonal system matrices in 2D. Again, the median spectral radius is always lower than 1.0. However, with an increasing problem size n, we see an increase of the number of outliers with a spectral radius $\rho > 1.0$.

In summary, on the analyzed problems, the iterative methods found by GPLS for small system matrices are generalizable and can be re-used for larger n, if the basic structure of the problem stays the same.

5.3 GPLS overcomes limitations of existing stationary iterative methods

The well-known Gauss–Seidel method converges if the system matrix A is either symmetric positive definite or strictly diagonally dominant. If this is not the case, there is no guarantee that the Gauss–Seidel method will find an appropriate iteration

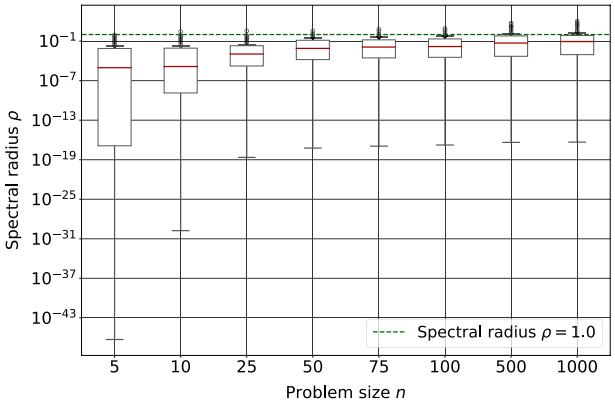


Fig. 6 Box-plots of the spectral radius ρ of diagonal 1D matrices over problem size n (starting with $n = 5$)

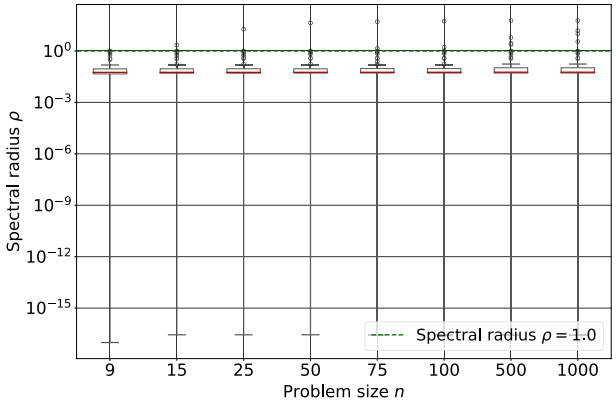


Fig. 7 Box-plots of the spectral radius ρ of diagonal 2D matrices over problem size n (starting with $n = 9$)

matrix G [7, 21]. To address such cases is a good challenge for GPLS because GP can search the whole space of potential methods and maybe come up with solutions for problems where the Gauss–Seidel method fails.

Consequently, we generate typical random system matrices where the Gauss–Seidel method cannot find an appropriate iteration matrix and study the properties of iteration matrices generated by GPLS. We use heat maps for the visual inspection of system and iteration matrices, which are graphical representations of the numerical elements in a matrix. Heat maps make it easier to see structural characteristics like diagonals and the sparsity of a matrix, as each entry/value is represented by a specific color.

Figure 8 shows a randomly generated dense system matrix of size 25×25. For this example, we filled the matrix with equally distributed integer values ranging from -10 to 10. The Gauss–Seidel method only finds an iteration matrix with a spectral radius of around 28,000. Hence, the Gauss–Seidel method cannot be used. In contrast, GPLS finds a solution for this example described by the term $(((AD) + ((U + D) + (L + D))) - (((D^{-1} + U) + (((L + D)^{-1} - U) - (D^{-1} + (L + D)^{-1}))) + ((AD) + ((U + D) + (L + D)))))$. Figure 9 shows the resulting

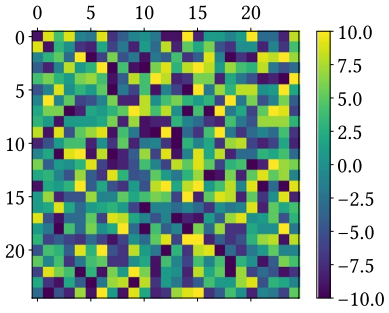


Fig. 8 Randomly generated dense system matrix

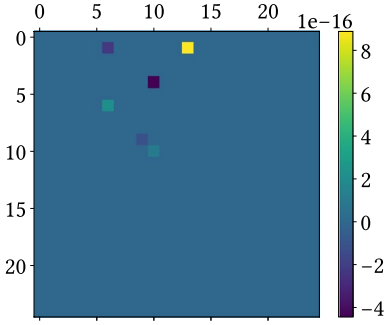


Fig. 9 Corresponding iteration matrix found by GPLS

iteration matrix. The matrix has a spectral radius of 2.22e−16 as well as high sparsity. The few non-zero values are concentrated in the upper triangular area because the found term is dominated by the terminals $L + D$ and U.

A second example is a randomly generated tridiagonal band matrix of size 25×25 as system matrix. For each diagonal, we used an equally distributed random integer value from the interval $[-10, 10]$. Thus, the band matrix is not diagonally dominant. Figure 10 shows the heat map for this system matrix. The spectral radius of the iteration matrix found by the Gauss–Seidel method is 6.0. Thus, the Gauss–Seidel method is not usable in this case.

In contrast, GPLS again finds an expression that is able to solve the problem. The term found by GPLS is $U + D^{-1}$. The resulting iteration matrix (see Fig. 11) has a spectral radius of 0.2 and is similar to the system matrix but has one diagonal less.

5.4 Convergence analysis of iteration matrices found by GPLS

This section studies the convergence speed of the iterative numerical methods found by GPLS for two types of dominant band matrices. We compare the solutions found by GPLS with those of the Jacobi, Gauss–Seidel, and SOR methods. For this purpose, we consider linear systems of the form

$$Ax = 0. \tag{5.1}$$

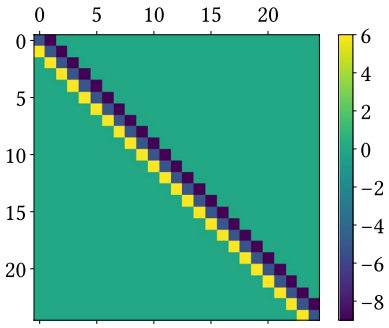


Fig. 10 Randomly generated not diagonally dominant band matrix as system matrix

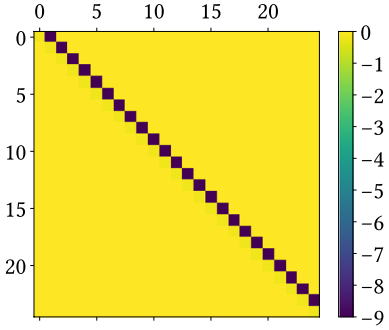


Fig. 11 Corresponding iteration matrix found by GPLS

5.4.1 Sparse diagonally dominant band matrices

In a first set of experiments, we study the convergence behavior for linear equations that arise from the discretization of PDEs. In particular, we consider Poisson's equation in 1D, 2D, and 3D with the following boundary condition (Dirichlet):

$$u(x) = 0.$$

We transform the PDEs into a system of linear equations (compare Sect. 3) using FDM, which leads to a system of the form of Eq. 5.1. In all three cases (1D, 2D, and 3D), the resulting system matrices are sparse diagonally dominant band matrices, for which, e.g., Jacobi and Gauss–Seidel are guaranteed to converge. GPLS evolved the following terms to calculate the iteration matrix G:

- 1D: $(D^{-1})^{13}((L+D)^{-1})^2U^6(D^{-1}+U)^3$
- 2D: $(D^{-1})^4(U-D^{-1})$
- 3D: $U+D^{-1}$

Table 2 compares the spectral radii of the iteration matrices of the Jacobi, Gauss–Seidel, SOR and GPLS method for all three cases of the discretized Poisson equation. For SOR, we set the relaxation parameter $\omega = 0.8$ [we tested values from the interval (0,2) with step size 0.1]. As expected ρ is lowest for iteration matrices found by GPLS. The spectral radii of iteration matrices constructed by the Jacobi or Gauss–Seidel method are only slightly lower than one.

To study the convergence behavior of the resulting iterative methods more closely, we employ the iteration scheme

Table 2 Spectral radii of the iteration matrices for the discretized Poisson equations

Case	Matrix size	Jacobi	Gauss–Seidel	SOR ($\omega = 0.8$)	GPLS
1D	175×175	0.99984	0.99968	0.23022	$1.03204e-7$
2D	196×196	0.97815	0.95677	0.2	0.00098
3D	216×216	0.90097	0.81174	0.2	0.16666

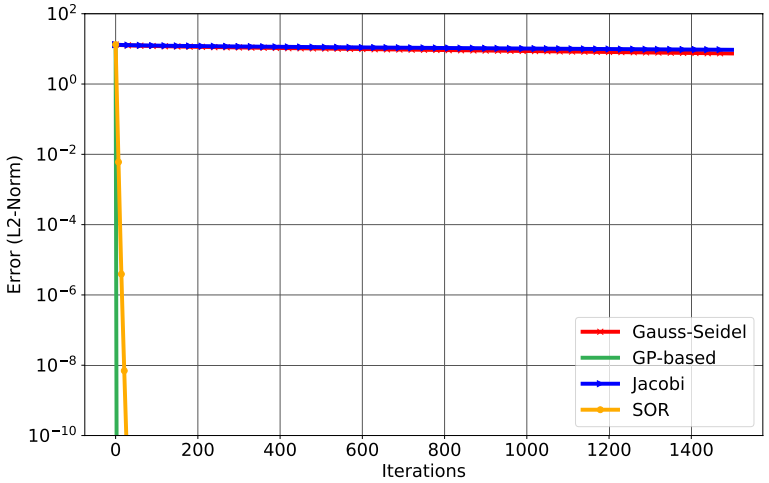
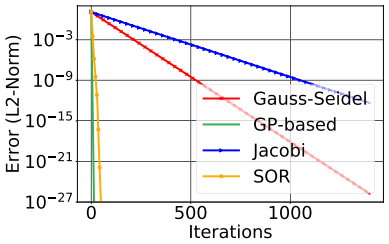


Fig. 12 Error over iterations for 1D Poisson

Fig. 13 Error over iterations for 2D Poisson



$$x^{(i+1)} = Gx^{(i)},$$

where $x^{(i)}$ is the current solution and G the iteration matrix. As initial guess $x^{(0)}$ for the solution of the system we use

$$x_j^{(0)} = 1 \quad \forall j = 1, \ldots n,$$

with n as the number of discretization points. As we know that the solution of the system defined in Eq. 5.1 is 0, the absolute error ϵ is equal to the current approximation $x^{(i)}$ during each iteration i:

$$\epsilon = x^{(i)} - 0 = x^{(i)}.$$

Figures 12, 13, and 14 plot the L^2-norm of the error ϵ over the number of iterations for the Jacobi, Gauss–Seidel, SOR, and GPLS-evolved iteration methods for the solution of Poisson's equation in 1D, 2D, and 3D, respectively. As expected, all three iteration schemes converge to the solution of the system, although—as reflected in the lower spectral radius of its iteration matrix—the iteration schemes

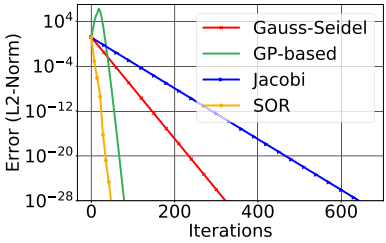


Fig. 14 Error over iterations for 3D Poisson

evolved by GPLS converge much faster than Gauss–Seidel and Jacobi. For example, in the 1D and 2D case convergence can be achieved with GPLS in only a few iterations. In the 3D case there is an increase of the error in the first few iterations followed by a fast decrease of the error. In all three instances, the convergence speed of SOR is similar to that of GPLS. However, the convergence speed strongly depends on the choice of the right relaxation parameter ω.

Being surprised by the extremely fast convergence of the iterative numerical methods evolved by GPLS (especially for the 1D case of Poisson's equation), we study whether GPLS has found as iteration matrix G the inverse of the system matrix A or a matrix that is very similar. If this is the case, the fast convergence behavior would be inevitable. Consequently, Fig. 15 shows the heat map of the product of A and the iteration matrix G found by GPLS. If the product would be the identity matrix I, then GPLS would have found A^{-1}. However, the figure shows that $AG \neq I$, because we have four diagonals in the upper triangular part of the matrix and no main diagonal.

5.4.2 Non-diagonally dominant band matrices

As a second and more challenging test case, we consider the class of non-diagonally dominant band matrices. For this class of matrices, e.g., the Jacobi and Gauss–Seidel methods are not guaranteed to converge in the general case. Thus, it is uncertain if a stationary iterative method that converges to the solution of an arbitrary linear system with a non-diagonally dominant system matrix can be evolved. To generate a

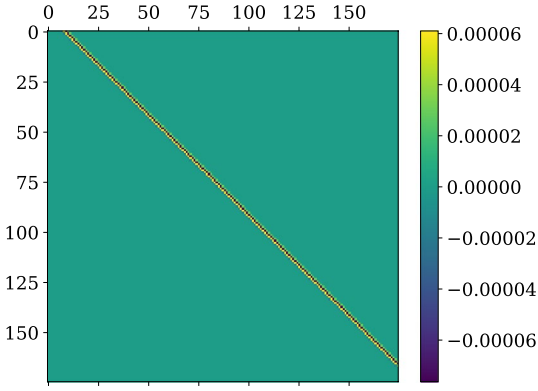


Fig. 15 Product of system matrix A and iteration matrix G for 1D Poisson

suitable instance of this class of matrices, we randomly generate a tridiagonal matrix of the form

$$A_1 = \begin{bmatrix} a & b & & & \\ c & a & b & & \\ & c & \ddots & \ddots & \\ & & \ddots & \ddots & b \\ & & & c & a \end{bmatrix},$$

that satisfies $|a| < |b| + |c|$. As a test case, we randomly choose the values $a = 4$, $b = 8$, and $c = 2$. We assume that this matrix corresponds to a one-dimensional problem. Thus, we can generate higher-dimensional problems of the same instance by computing the Kronecker sum of the matrix with itself:

$$A_2 = A_1 \oplus A_1,$$
$$A_3 = A_2 \oplus A_1.$$

The resulting system matrices are also non-diagonally dominant.

Table 3 shows the spectral radii of the resulting Jacobi, Gauss–Seidel, and SOR iteration matrices, as well as of the iteration matrices evolved by GPLS. For SOR, we set the relaxation parameter $\omega = 0.6$ [again, we tested values from the interval (0,2) with step size 0.1]. The spectral radii of the iteration matrices generated by Jacobi and Gauss–Seidel are all larger than one. Thus, convergence cannot be guaranteed. In contrast, SOR and GPLS can evolve iteration matrices with a spectral radius smaller than one. For the band matrices in 1D, 2D, and 3D, GPLS evolved the following terms to calculate the iteration matrix G:

- 1D: $(D^{-1})^2(D^{-1}U(D - A) + D)$
- 2D: $D^{-1} + U$
- 3D: $D^{-1} + U$

Analogous to the Poisson case, we study the convergence of the resulting iterative methods by solving the system defined in Eq. 5.1, using the same initial guess $x^{(0)} = 1$. Again, we measure the L^2 norm of the error ϵ compared to the exact solution 0 during each iteration.

Figures 16, 17, and 18 plot the error over the number of iterations. As expected, the Jacobi and Gauss–Seidel methods do not converge in any of the three cases,

Table 3 Spectral radii of the iteration matrices for a non-diagonally dominant band matrix

Case	Matrix size	Jacobi	Gauss–Seidel	SOR ($\omega = 0.6$)	GPLS
A_1	175×175	1.99949	3.99873	0.57392	0.25
A_2	196×196	1.9563	3.82709	0.4	0.125
A_3	125×125	1.73205	3.0	0.4	0.08333

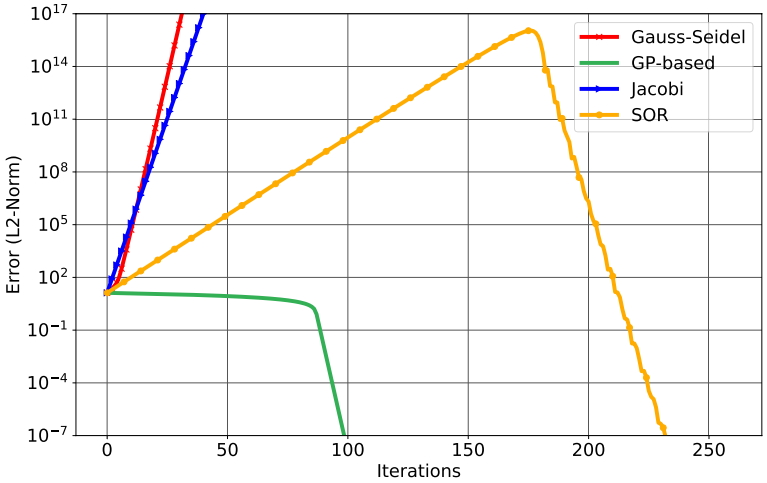


Fig. 16 Error over iterations for a non-diagonally dominant band matrix in 1D

Fig. 17 Error over iterations for a non-diagonally dominant band matrix in 2D

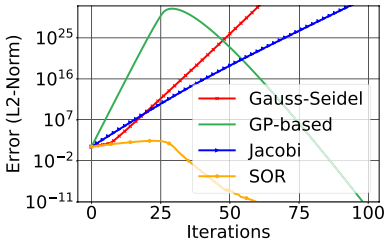
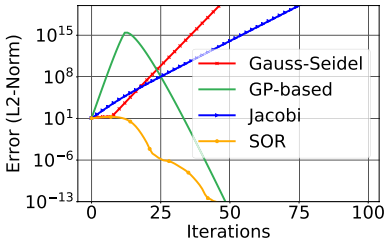


Fig. 18 Error over iterations for a non-diagonally dominant band matrix in 3D



but the error increases further during each iteration. In contrast, GPLS was able to evolve an iteration matrix that leads to convergence in all three cases. The convergence speed is on a level similar to the SOR method (in all three studied instances).

If we compare the convergence behavior of GPLS of non-diagonally dominant band matrices to the Poisson case (see Figs. 12, 13, and 14), we find that the evolved schemes on average require more iterations and that convergence is only achieved after an initial stagnation or even an increase of the error. Nevertheless, the evolved iteration matrices always lead to low errors in less than 100 iterations. The initial error increase can be explained by the fact that within a stationary iterative method,

not all error components can be eliminated simultaneously. Consequently, the reduction of certain error components can cause an increase in the remaining ones and, thus, lead to the observed overall growth of the approximation error. However, after this initial error increase, the total error quickly decreases (with GPLS and SOR), which means that after particular error components are eliminated within the first few iterations, the remaining ones are efficiently reducible.

6 Conclusions

Numerical methods are used to solve problems where no analytical solutions exist or are difficult to find. In many real-world applications, the number of unknowns is huge which makes efficient and scalable solvers for such systems necessary. As GP is known for finding human-competitive results for many real-world problems [18], its combination with domain knowledge from the classical numerical methods allows us to come up with iteration matrices that beat existing iterative numerical methods.

This paper proposed GPLS, a GP-based approach that searches for mathematical expressions that define iteration matrices for given linear systems. The found iteration matrices are used by stationary iterative methods which numerically solve the system of linear equations. GPLS makes use of the elements of existing methods – like variations of the system matrix—to find iteration matrices that lead to a fast and reliable convergence of iterative numerical methods. Additionally, GPLS finds iteration matrices that are sparse in its structure such that the resulting iterative numerical methods can be executed efficiently on parallel computer architectures.

The results show that GPLS finds iteration matrices with a low spectral radius for both, dense and also sparse diagonal system matrices. Furthermore, the found iteration matrices are of high sparsity and the mathematical term describing these matrices is often of low complexity (small parse tree). The found solutions are often generalizable to larger dimensions in the sense that solutions found for small problems also work well for larger system matrices. We showed this for two classes of band matrices as the terms found by GPLS for small system matrices ($n \leq 9$) can be often used to compute high quality iteration matrices with a spectral radius $\rho < 1.0$, even for larger problem instances (up to $n = 1000$).

We also found that GPLS can find solutions where the classical iterative methods (the Gauss–Seidel and the Jacobi methods) fail to find appropriate iteration matrices. Furthermore, the iterative methods found by GPLS converge much faster compared to the Gauss–Seidel and Jacobi methods on the studied test problems and perform like the SOR method but without the need of an additional relaxation parameter.

7 Future work

In this work, we demonstrated that GPLS can evolve effective stationary iterative methods for solving different sparse linear systems. Another direction is the use of these methods for the preconditioning of Krylov subspace methods [21]. In this

case, the goal is not to directly solve a sparse linear system but, instead, to use a stationary iterative method to compute an approximation for the inverse of a preconditioning matrix P. This approximation is then applied to the original system A, for instance, to obtain a right-preconditioned system $AP^{-1}u = b$, which is easier to solve than the original system $Ax = b$.

So in future work, we will study the ability of GPLS to evolve optimal stationary iterative methods to solve systems of the form $Px = u$, where the solution x represents an approximation for $P^{-1}u$. The evolved method can then easily be integrated into an existing solver for the resulting preconditioned system, such as a Krylov subspace method, to evaluate its effectiveness on different test cases.

Additionally, we will further analyze the scalability/generalizability of solutions found by GPLS and study ways to approximate the spectral radius—as a quality indicator for iteration matrices—and find other problem representations to enable an even faster computation.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. W.F. Ames, *Numerical Methods for Partial Differential Equations* (Academic press, 2014)
2. A. Amritkar et al., Recycling Krylov subspaces for CFD applications and a new hybrid recycling solver. J. Comput. Phys. **303**, 222–237 (2015)
3. R. Barrett et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Vol. 43 (SIAM, 1994)
4. D.S. Burke et al., Putting more genetics into genetic algorithms, in Evolutionary Computation 6.4 (1998), pp. 387–410
5. T.A. Davis, *Direct Methods for Sparse Linear Systems*, Vol. 2 (Society of Industrial and Applied Mathematics, 2006)
6. J.W. Demmel, *Applied Numerical Linear Algebra*, Vol. 56 (Society of Industrial and Applied Mathematics, 1997)
7. G.H. Golub, C.F. Van Loan, *Matrix Computations*, Vol. 3. (HU Press, 2012)
8. R.M. Gray, in Toeplitz and circulant matrices: a review (2006)
9. R.A. Horn, C.R. Johnson, *Matrix Analysis*, 2nd edn. (Cambridge University Press, Cambridge, 2012)
10. C. Johnson, *Numerical Solution of Partial Differential Equations by the Finite Element Method* (Courier Corporation, 2012)
11. W.M. Kahan, in Gauss–Seidel methods of solving large systems of linear equations (2002)
12. M. Keijzer, Improving symbolic regression with interval arithmetic and linear scaling, in European Conference on Genetic Programming (Springer, 2003), pp. 70–82

13. J.R. Koza, *Genetic Programming Ii: Automatic Discovery of Reusable Programs* (MIT Press, 1994)
14. J.R Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, 1992)
15. R.G. Mahmoodabadi, H. Köstler, Genetic Programming Meets Linear Algebra: How Genetic Programming Can Be Used to Find Improved Iterative Numerical Methods, in Proceedings of the Genetic and Evolutionary Computation Conference Companion. GECCO '17. Berlin, Germany: ACM (2017), pp. 1403–1406
16. D.J. Montana, Strongly typed genetic programming, in Evolutionary computation 3.2 (1995), pp. 199–230
17. K.W. Morton, D.F. Mayers, *Numerical Solution of Partial Differential Equations: An Introduction* (Cambridge university press, 2005)
18. R. Poli, W.B. Langdon, Nicholas Freitag McPhee. A field guide to genetic programming. (With contributions by J. R. Koza). Published via http://lulu.com (2008)
19. R. Poli, Nicholas Freitag McPhee. Parsimony Pressure Made Easy, in Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation. GECCO '08. Atlanta, GA, USA: ACM (2008), pp. 1267–1274
20. H. Rittich, Extending and Automating Fourier Analysis for Multigrid Methods. Ph.D. thesis, University of Wuppertal, June (2017)
21. Y. Saad, *Iterative Methods for Sparse Linear Systems*, Vol. 82 (Society of Industrial and Applied Mathematics, 2003)
22. J. Schmitt, S. Kuckuk, H. Köstler, Constructing Efficient Multigrid Solvers with Genetic Programming, in Proceedings of the 2020 Genetic and Evolutionary Computation Conference. GECCO '20. Cancún, Mexico: ACM (2020), pp. 1012–1020
23. G.D. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods* (Oxford university press, 1985)
24. R. Temam, *Navier–Stokes Equations: Theory and Numerical Analysis*, Vol. 343 (American Mathematical Soc., 2001)
25. L.N. Trefethen, D. Bau III, *Numerical Linear Algebra*, Vol. 50 (Society of Industrial and Applied Mathematics, 1997)
26. H.K. Versteeg, W. Malalasekera, *An Introduction to Computational Fluid Dynamics: The Finite Volume Method* (Pearson Education, 2007)
27. E.J. Vladislavleva, G.F. Smits, D.D. Hertog, Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming, in IEEE Transactions on Evolutionary Computation 13.2 (2008), pp. 333–349
28. R. Wienands, W. Joppich, *Practical Fourier Analysis for Multigrid Methods* (CRC Press, 2004)
29. D.M. Young, *Iterative Solution of Large Linear Systems* (Elsevier, 2014)