PARALLEL CONJUGATE GRADIENTS ON SPARSE STIFFNESS MATRICES

RAYMOND VAN VENETIË AND JAN WESTERDIEP

1. Introduction

> "The conjugate gradient method (CG) is an algorithm for the numerical solutions of particular systems of linear equations, namely those whose matrix is symmetric and positive definite. The conjugate gradient method is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation or other direct methods such as the Cholesky decomposition. Large sparse systems often arise when numerically solving partial differential equations or optimization problems." [Wik14a]
>
> "The finite element method (FEM) is a numerical technique for finding approximate solutions to boundary value problems for partial differential equations. It uses subdivision of a whole problem domain into simpler parts, called finite elements, and variational methods from the calculus of variations to solve the problem by minimizing an associated error function. Analogous to the idea that connecting many tiny straight lines can approximate a larger circle, FEM encompasses methods for connecting many simple element equations over many small subdomains, named finite elements, to approximate a more complex equation over a larger domain." [Wik14b]

We will be concerned with solving

$$Ax = b$$

with known A and b.

In this report, we will look at a parallel implementation of the conjuate gradient method. We will solve linear systems coming from the finite element method. Systems like this are sparse in the sense that a lot of the elements will be zero. This makes it possible to skip parts of the matrix-vector products one encounters.

To test our implementation, it is useful to have a supply of matrices ready. The University of Florida has a huge collection of sparse matrices. [DH14] We chose to read these matrices in Matrix Market file format[1]. [NIS07]

2. Finite Element Method

Given the boundary value problem,

$$\begin{cases} -\Delta u = 1 & \Omega \\ u = 0 & \partial\Omega \end{cases} \tag{1}$$

{eqn:problem}

and some partition into simplices, FEM will find the continuous solution, piecewise polynomial (wrt. this partition) with smallest H^1-norm error. ofzo

[1]The reason for this is pure laziness: it is possible to download these matrices in this format and the creators of the Matrix Market supplies a I/O C library.

If we apply this to our 2D case, we have a polygonal domain Ω and a set of elements $\{\triangle_k\}_{k=1}^K$, $\triangle_k \subset \Omega$ with $\cup_{k=1}^K \triangle_k = \Omega$ and $\triangle_k \cap \triangle_j$ for $k \neq j$ is either empty, a common vertex or a common edge (hoe heet dit ook alweer? regularity condition ofzo?).

We create one reference element $\hat{\triangle}$ spanned by vertices $(0,0)$, $(1,0)$, $(0,1)$ on which we have a polynomial basis of some degree – say $\bar{p}$. The amount of basis functions in this basis must be $p = (\bar{p}+2)(\bar{p}+1)/2$. So we have a basis $\hat{\Phi} = \{\hat{\phi}_i\}_{i=1}^p$.

We are now able to create an affine function $T_k : \triangle_k \to \hat{\triangle}$ from some element in the partition to this reference element. Hence we automatically have a polynomial basis on $\triangle_k$, namely $\Phi^k := \{\phi_i^k\}_{i=1}^p$ with $\phi_i^k := \hat{\phi}_i \circ T_k$.

We can in fact create a global basis $\mathbf{\Phi}$ by gluing together the correct functions (TODO dit is lastig). Each basis function of this basis is a piecewise polynoimal subject to the partition, and is necessarily zero on $\partial\Omega$.

2.1. **Weak formulation.** If u solves (1), then

$$-\Delta u = 1 \implies -v\Delta u = v \forall v \in H_0^1(\Omega) \implies \int_\Omega -v\Delta u = \int_\Delta v$$

and using Green's first identity

$$\int_\Omega -v\Delta u = \int_\Omega \nabla v \cdot \nabla u - \oint_{\partial\Omega} v(\nabla u \cdot n)$$

where the last term must equal zero as $u = 0$ on $\partial\Omega$. We therefore end up at the weak formulation:

$$u \text{ solves (1)} \implies \int_\Omega \nabla v \cdot \nabla u = \int_\Omega v \forall v \in H_0^1(\Omega).$$

The idea is that we will find functions u_{FE} that exhibit this property and "hope" (TODO vinden dat dit klopt) that u_{FE} "almost" solves (1).

3. Linear polynomials on the triangle

If we take $\bar{p} = 1$, we are discussing linear polynomials on the triangle. One way to create a basis for this space is to use the *nodal basis*, where we create basis functions that are equal 1 on one vertex of the triangle and 0 on the others. The nodal basis on the reference element becomes

$$\hat{\phi}_1 = 1 - y - x, \quad \hat{\phi}_2 = y, \quad \hat{\phi}_3 = x.$$

If we look at this globally, we have a *nodal basis* for the whole partition, namely two-dimensional "hat" functions ϕ_i which are zero on every vertex but a single one – say v_i. The vertices on the boundary of the domain cannot have basis functions associated with them, as the resulting solution $u_{FE} = c^\top \mathbf{\Phi} = \sum c_i \phi_i$ must be zero on this boundary.

If we fill in u_{FE} in this weak formulation and set $v = \phi_j$, we get

$$\int_\Omega \nabla\phi_j \cdot \nabla\left(\sum c_i\phi_i\right) = \int_\Omega \phi_j \implies \sum c_i \int_\Omega \nabla\phi_j \cdot \nabla\phi_i = \int_\Omega \phi_j \forall j$$

or

$$Ac = b, \quad a_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i, \quad b_i = \int_\Omega \phi_j.$$

In other words, finding this best solution $u_{FE} = c^\top \mathbf{\Phi}$ amounts to solving a linear system where the matrix A is real and symmetric (as inner products are commutative). This is where the Conjugate gradient method comes into play.

Hier moet nog aan toegevoegd worden:

- stiffness matrix is eigenlijk som van kleinere stiffness matrices
- deze grote stiffness matrix kan je maken/opslaan door op de goede plek shit in te voegen.

4. CG

In its most basic (sequential, non-preconditioned) form, CG can be written down as in Algorithm 1. [Bis04, Alg. 4.8] We slightly adapted it from its original form to be able to use the different BLAS routines.

4.1. **Sparse CG.**

As FEM matrices are sparse in nature, we will adapt Algorithm 1 to a version that supports sparse matrices. For simplicity, we will assume the right-hand side to be dense. This allows us to effectively only change I/O and the function responsible for matrix-vector multiplication.

Our storage format uses the so-called coordinate scheme; we store tuples (i, j, a_{ij}) with i the row number, j the column number and a_{ij} the matrix value at this position. The Matrix Market file format adds some headers, e.g. to denounce symmetry so that one only has to store the lower triangular part. We denote by $nz(A)$ the amount of nonzero elements in this lower triangular part. If we store the list of tuples in three lists of length $nz(A)$, namely I, J and v, we can compute a sequential sparse matrix-vector multiplication using Algorithm 2 (from [Bis04, Alg. 4.3]).

4.2. **Preconditioned CG.**

The iterates x_k obtained from the CG algorithm satisfy the following inequality [Sle14, Slide 23]:

$$\frac{\|x - x_k\|_A}{\|x - x_0\|_A} \leq 2\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right)^k \leq 2\exp\left(-\frac{2k}{\sqrt{\kappa_2(A)}}\right)$$

where $\kappa_2(A)$ is the 2-condition number of A, which for symmetric positive definite matrices equates

$$\kappa_2(A) = \frac{\lambda_{max}}{\lambda_{min}}.$$

It is therefore of interest to create a condition number that is as low as possible.

A preconditioner P of a matrix A is a matrix such that $P^{-1}A$ has a smaller condition number than A. As the theoretical convergence rate is highly dependent on the condition number, we can improve this using such a preconditioner. Instead of solving $Ax = b$, we will solve $P^{-1}Ax = P^{-1}b$. Blablabla

5. Parallellizing CG

6. Using sparsity of matrices in parallel CG

7. hoe heette het ook alweer als je na een kleine verandering in je matrix opnieuw ging optimizen

8. Future work

8.1. **Adaptive FEM.**

References

[Bis04] Rob H. Bisseling. *Parallel Scientific Computation: A Structured Approach using BSP and MPI*. Oxford University Press, Oxford, UK, March 2004.

[DH14] Tim Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. `http://www.cise.ufl.edu/research/sparse/matrices`, 2014.

[NIS07] NIST. Matrix Market. `http://math.nist.gov/MatrixMarket`, 2007.

Algorithm 1 Sequential CG

Require: $k_{max} \in \mathbb{N}$, $\epsilon \in \mathbb{R}$, $n \in \mathbb{N}$, A symmetric $n \times n$, $b \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$
Ensure: $x \in \mathbb{R}^n$ with $Ax \approx b$

```
1: int k := 0;
2: float r[n];
3: float ρ;
4: float ρ_old;
5: float nbsq;
6:
7: x ← x_0;
8: r ← b;
9: r ← r − Ax;
10: ρ ← ⟨r, r⟩;
11: nbsq ← ⟨b, b⟩;
12:
13: while ρ > ε² · nbsq ∧ k < k_max do
14:     float p[n];
15:     float w[n];
16:     float α;
17:     float β;
18:     float γ;
19:
20:     if k == 0 then
21:         p ← r;
22:     else
23:         β ← ρ/ρ_old;
24:         p ← βp;
25:         p ← r + p;
26:     end if
27:
28:     w ← Ap;
29:     γ ← ⟨p, w⟩;
30:     α ← ρ/γ;
31:     x ← x + αp;
32:     r ← r − αw;
33:     ρ_old ← ρ;
34:     ρ ← ⟨r, r⟩;
35:
36:     k ← k + 1;
37: end while
```

[Sle14] Gerard L.G. Sleijpen. Numerical linear algebra — krylov methods for hermitian systems. `http://www.staff.science.uu.nl/~sleij101/Opgaven/NumLinAlg/Transparancies/Lecture7.pdf`, 2014.

[Wik14a] Wikipedia. Conjugate gradient method — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Conjugate_gradient_method`, 2014.

[Wik14b] Wikipedia. Finite element method — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Finite_element_method`, 2014.

Algorithm 2 Sequential sparse matrix vector multiplication: find $y \leftarrow \alpha Ax + \beta y$ for symmetric A

Require: $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}$, $n \in \mathbb{N}$, amount of nonzeroes $nz(A)$, $I \in \mathbb{N}^{nz(A)}$, $J \in \mathbb{N}^{nz(Z)}$, $v \in \mathbb{R}^{nz(A)}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^n$

Ensure: $y \leftarrow \alpha Ax + \beta y$

1: **for** $i = 0$ until n **do**
2: $\quad y[i] = \beta y[i]$;
3: **end for**
4: **for** $j = 0$ until $nz(A)$ **do**
5: $\quad y[I[j]] = \alpha v[j] x[J[j]] + y[I[j]]$;
6: $\quad$ **if** $I[j] \neq J[j]$ **then** $\quad \triangleright$ A is symmetric
7: $\quad\quad y[J[j]] = \alpha v[j] x[I[j]] + y[J[j]]$;
8: $\quad$ **end if**
9: **end for**