Krylov Methods

A (very) incomplete introduction

Riccardo Rossi

Real basics…

Let’s begin with a question: imagine that i have a given vector

$$\boldsymbol{r}(\boldsymbol{x}) \coloneqq \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$$

What does it mean to require $\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{0}$?

Real basics…

Let’s begin with a question: imagine that i have a given vector (of size N)

$$\boldsymbol{r}(\boldsymbol{x}) \coloneqq \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$$

What does it mean to require $\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{0}$?

ANSWER – it means that all of its components are zero…

…something that we can write as

$$\boldsymbol{r}(\boldsymbol{x}) \cdot \boldsymbol{e}_i = \boldsymbol{0} \quad \forall \boldsymbol{i}$$

THAT IS: a vector is zero if it is ortogonal to all of the vectors of a basis of the space it lives in

Real basics…

But which basis? **ANY !!**

THAT IS: if $\{v_0 \dots v_N\}$ form a basis of R^N than a vector is zero if and only if $\boldsymbol{r}(\boldsymbol{x}) \cdot \boldsymbol{v}_i = \boldsymbol{0} \quad \forall \boldsymbol{i}$

IN A NUTSHELL, Krylov methods are all about **constructing a basis** (that grows with the number of iterations until spanning the entire R^N) and **making the residual to be Orthogonal to such basis**.

Krylov Subspace

Of course we have complete freedom of “incrementally constructing” a space that eventually describes the entire R^N.

The space employed by Krylov techniques is known as “**Krylov Subspace**”, defined as

$$K_i(\boldsymbol{A},\boldsymbol{b}) = span\{\boldsymbol{b},\boldsymbol{Ab},\boldsymbol{A^2b} \ldots, \boldsymbol{A^i b}\}$$

Provided that the matrix A is invertible, and that b has a component wrt all of the eigenvectors of A, such space will eventually grow with the iterations until coinciding with R^n

Krylov techniques differe of the construction and properties of a basis of $K_i(A, b)$.

A property shared by all of the methods in the Krylov family is that **they will converge** (in exact algebra) **in at most N iterations** (although they will typically converge way before)

Minimizing residual along a line

A tool very frequently used, is the **minimization of the residual along a given direction**. Let's imagine that we have a starting solution $\boldsymbol{x}_0$ and a search direction identified by a unit vector $\boldsymbol{v}$

The idea is to choose a new $\boldsymbol{x} \coloneqq \boldsymbol{x_0} + \boldsymbol{\alpha v}$ such that $||\boldsymbol{r}(\boldsymbol{x})||^{\boldsymbol{2}}$ is minimal in a choosen norm. One way to accomplish this, is to make the residual to be ortogonal to the direction $\boldsymbol{v}$, that is, to require that $\boldsymbol{v} \cdot \boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{0}$

Using the definition we get

$$0 = \boldsymbol{v} \cdot \boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{v} \cdot (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x_0} - \boldsymbol{\alpha A v}) = \boldsymbol{v} \cdot (\boldsymbol{r}(\boldsymbol{x_0}) - \boldsymbol{\alpha A v})$$

Solving for alpha

$$\boldsymbol{\alpha} = \frac{\boldsymbol{v} \cdot \boldsymbol{r}(\boldsymbol{x_0})}{\boldsymbol{v} \cdot \boldsymbol{A v}}$$

Special Case of SPD matrices

If $\boldsymbol{A}$ is SPD, we can define a functional $\boldsymbol{\Psi}(\boldsymbol{x}) := \boldsymbol{x}^t\boldsymbol{b} - \frac{1}{2}\boldsymbol{x}^t Ax$ (which we will use both for CG and SD)

It is easy to see that:

1 such that $\boldsymbol{Ax_{ex}} = \boldsymbol{b}$ *is the (only) minimum* of $\boldsymbol{\Psi}$ (easy to prove since $\mathbf{A}$ is SPD, hence $\boldsymbol{v}^t\boldsymbol{A}\boldsymbol{v} > 0$ for any non zero $\mathbf{v}$)

2 The gradient of the function is $\boldsymbol{\nabla\Psi} = \boldsymbol{b} - \boldsymbol{Ax} = \boldsymbol{r}(\boldsymbol{x})$ (obviously zero in $\boldsymbol{\nabla\Psi}\ (\boldsymbol{x}_{ex}) = \boldsymbol{0}$)

Steepest Descent

(not a member of the Krylov Family)

The "Steepest Descent" Is the first idea one may have. It Works as follows:

1 choose a starting point $\boldsymbol{x_0}$ and use $\nabla\Psi$ as search direction

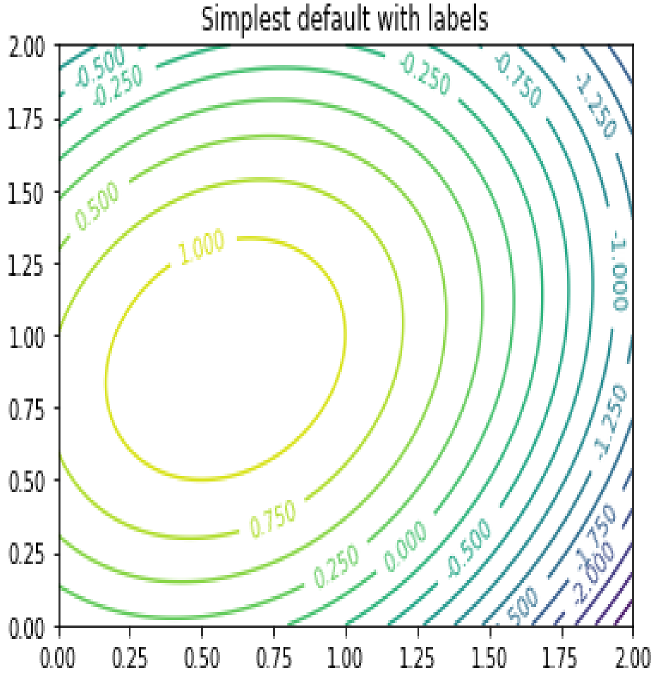
$$\boldsymbol{v} := \frac{\nabla\Psi\,(\boldsymbol{x}_0)}{\|\nabla\Psi(\boldsymbol{x}_0)\|} = \frac{\boldsymbol{r}(\boldsymbol{x}_0)}{\|\boldsymbol{r}(\boldsymbol{x}_0)\|}$$

2 evaluate the minimum in the direction of $\boldsymbol{v}$ starting from $\boldsymbol{x}_0$, that is, compute $\boldsymbol{x_1} = \boldsymbol{x_0} + \boldsymbol{\alpha}\,\boldsymbol{v}$ (with $\boldsymbol{\alpha} := -\frac{\boldsymbol{v}^t\boldsymbol{r}(\boldsymbol{x}_0)}{\boldsymbol{v}^t A\boldsymbol{v}}$, using the minimization formula in the previous slides)

3 repeat until $\frac{\boldsymbol{r}(\boldsymbol{x}_i)}{\|\boldsymbol{b}\|} < \epsilon$

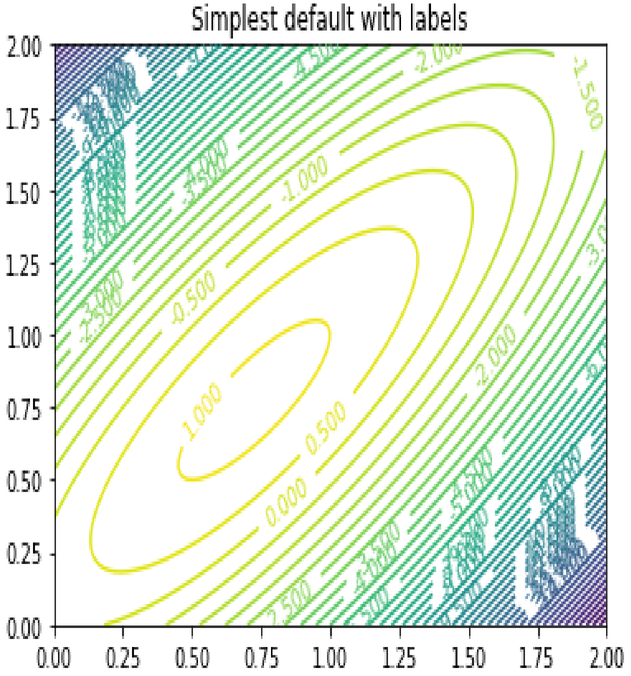
SIMPLE but … **may be very slow** (no guarantees it converges in k)

Why it takes long?



Plot of $\Psi(x) = \boldsymbol{x^t b} - 1/2\boldsymbol{x^t A x}$

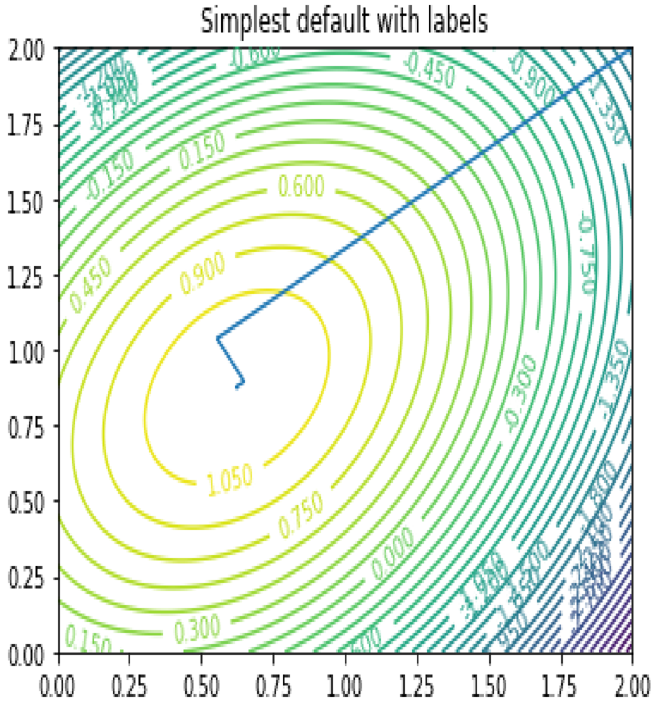
$A = \begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad k(A) = 2$



Plot of $\Psi(x) = \boldsymbol{x^t b} - 1/2\boldsymbol{x^t A x}$

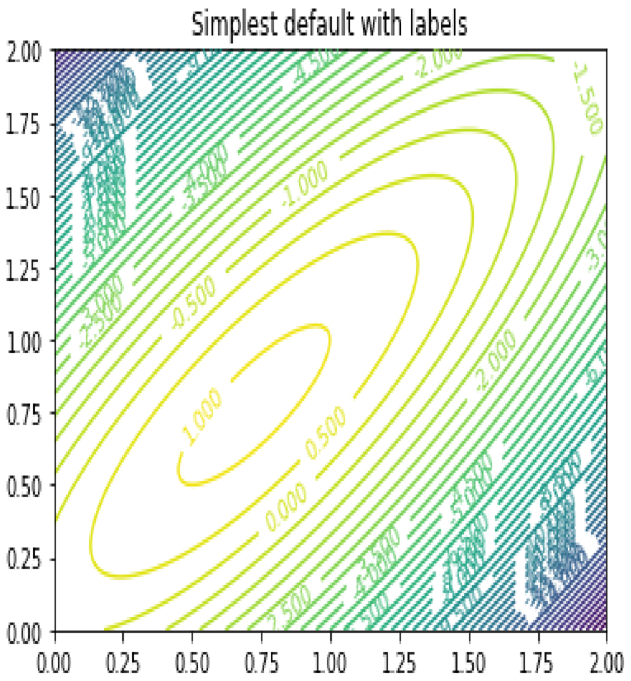
$A = \begin{pmatrix} 11 & -9 \\ -9 & 11 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad k(A) = 10$

Why it takes long?



$Plot\ of\ \Psi(x) = \boldsymbol{x^t b} - 1/2\boldsymbol{x^t A x}$

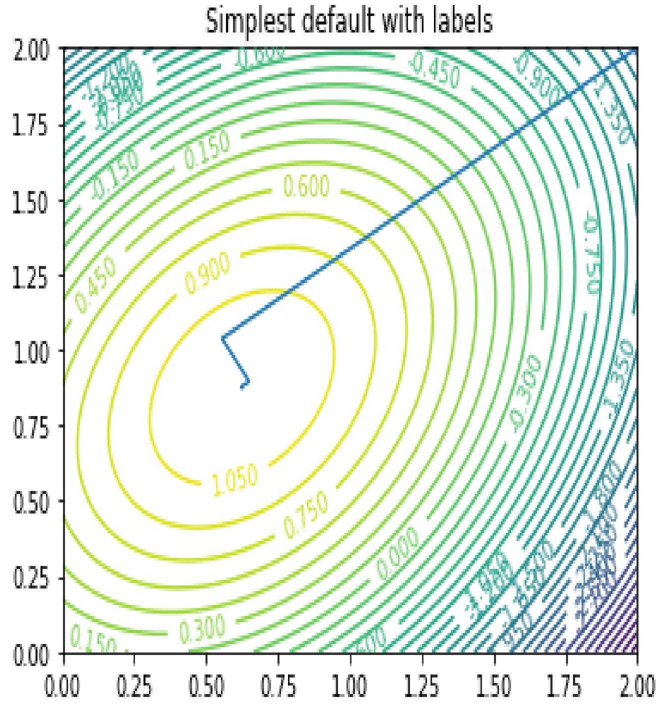
$A = \begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad k(A) = 2$



$Plot\ of\ \Psi(x) = \boldsymbol{x^t b} - 1/2\boldsymbol{x^t A x}$

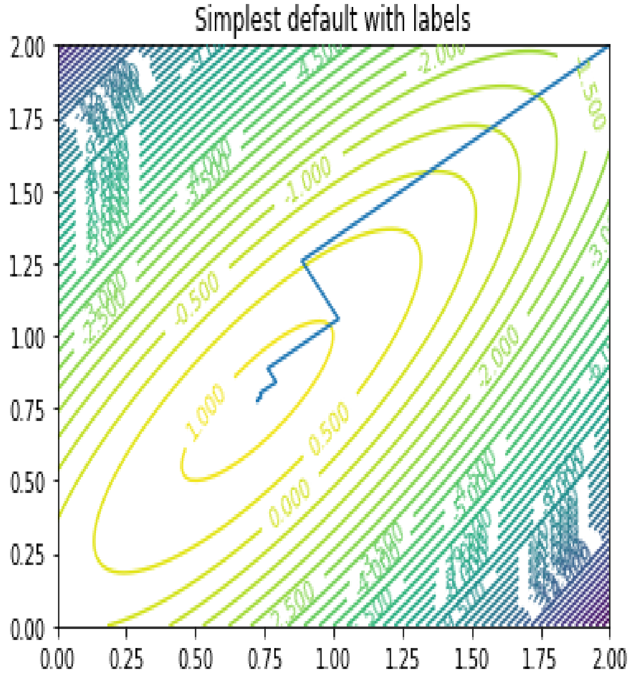
$A = \begin{pmatrix} 11 & -9 \\ -9 & 11 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad k(A) = 10$

Why it takes long?



Plot of $\Psi(x) = \boldsymbol{x^t b} - 1/2\boldsymbol{x^t A x}$

$$A = \begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad k(A) = 2$$



Plot of $\Psi(x) = \boldsymbol{x^t b} - 1/2\boldsymbol{x^t A x}$

$$A = \begin{pmatrix} 11 & -9 \\ -9 & 11 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad k(A) = 10$$

Conjugate Gradient

The “Conjugate Gradient” method can be understood as an improvement over the “steepest descent”. It Works as follows:

1 - choose a starting point $\boldsymbol{x_0}$ and use $\boldsymbol{v_0} = \boldsymbol{r}(\boldsymbol{x_0}) = \boldsymbol{r_0}$ as first search direction (**as the steepest descent!**)

2 $\boldsymbol{x_i} = \boldsymbol{x_0}, \boldsymbol{v_i} = \boldsymbol{v_0}$

3 - $\boldsymbol{x_{i+1}} = \boldsymbol{x_i} + \boldsymbol{\alpha}\, \boldsymbol{v_i}$ (with $\boldsymbol{\alpha} := -\frac{\boldsymbol{v_0}^t \boldsymbol{r}(\boldsymbol{x_0})}{\boldsymbol{v_0}^t \boldsymbol{A} \boldsymbol{v_0}}$)

4 - choose a new update direction as $\boldsymbol{v_{i+1}} := \boldsymbol{r}(\boldsymbol{x_{i+1}}) + \sum_{k=0}^{i} \beta_{ik} \boldsymbol{v}_k$

where the β_{ik} are chosen so that $\boldsymbol{A}\boldsymbol{v_{i+1}} \cdot \boldsymbol{v_k} = \boldsymbol{0} \quad \forall \boldsymbol{k}$

(**A-orthogonality instead of orthogonality!!**)

5 go back to 3, until $\frac{\boldsymbol{r}(\boldsymbol{x_{i+1}})}{\|\boldsymbol{b}\|} < \epsilon$

The key difference wrt the steepest descent is in Step 4, in the choice of the

A-orthogonality

Let's focus on the orthogonalization step, taking in mind the equation $\boldsymbol{v_{i+1}}{:=}\boldsymbol{r}(\boldsymbol{x_{i+1}}) + \sum_{k=0}^{i} \beta_{ik}\boldsymbol{v}_k$:

1 – FIRST ITERATION $\boldsymbol{v_1}{:=}\boldsymbol{r}_1 + \beta_{00}\boldsymbol{v}_0$ A-orthogonality :

$$\boldsymbol{v_0^t A v_1 = 0 \Rightarrow v_0^t A r_1 + \beta_{10} v_0^t A v_0 = 0 \Rightarrow \beta_{10} = -\frac{v_0^t A r_1}{v_0^t A v_0}}$$

2 – SECOND ITERATION $\boldsymbol{v_2}{:=}\boldsymbol{r}_2 + \beta_{20}\boldsymbol{v}_0 + \beta_{21}\boldsymbol{v}_1$ A-orthogonality:

$$\boldsymbol{v_0^t A v_2 = 0 \Rightarrow v_0^t A r_2 + \beta_{20} v_0^t A v_0} + \beta_{21} v_0 A v_1 \boldsymbol{= 0 \Rightarrow \beta_{20} = -\frac{v_0^t A r_2}{v_0^t A v_0^t}}$$

$$\boldsymbol{v_1^t A v_2 = 0 \Rightarrow v_1^t A r_2} + \beta_{20} \boldsymbol{v_1^t} A v_0 + \boldsymbol{\beta_{21} v_1^t A v_1 = 0 \Rightarrow \beta_{21} = -\frac{v_1^t A r_2}{v_1^t A v_1}}$$

note that we used here that $v_0 A v_1 = 0$ and that **A is symmetric**, hence $v_0 A v_1 = v_1 A v_0 = 0$.

Also, **the terms in the denominator are guaranteed to be positive since A is SPD**

3 – OTHER ITERATIONS:

$$\boldsymbol{v_{i+1}} \coloneqq \boldsymbol{r_{i+1}} + \sum_{k=0}^{i} \beta_{ik}\boldsymbol{v}_k \text{ with } \boldsymbol{\beta_{(i+1)k} = -\frac{v_k^t A r_{i+1}}{v_k^t A v_k} \forall k \le i}$$

Conjugate Gradient Magic

Let's make some observations:

- by construction $\boldsymbol{r}_{i+1} \cdot \boldsymbol{v}_i = 0$ (that's how we choose α).
- It is easy to prove that $\boldsymbol{r}_{i+1} \cdot \boldsymbol{v}_i = 0 \Rightarrow \boldsymbol{r}_{i+1} \cdot \boldsymbol{v}_j = 0 \;\; \forall \, \mathrm{j} \leq i$
- The last sentence can be paraphrased as follows:

$$\boldsymbol{r}_{i+1} \perp \text{span}\{\boldsymbol{v}_0 \ldots \boldsymbol{v}_i\}$$

- now, $\boldsymbol{v}_{\boldsymbol{i+1}} = \boldsymbol{r}(\boldsymbol{x}_{\boldsymbol{i+1}}) + \sum_{k=0}^{i} \beta_{ik} \boldsymbol{v}_k$ hence $\boldsymbol{v}_{\boldsymbol{i+1}}$ **is a linear combination of the previous residuals(*)**. It follows that

$$\text{span}\{\boldsymbol{v}_0 \ldots \boldsymbol{v}_i\} = \text{span}\{\boldsymbol{r}_0 \ldots \boldsymbol{r}_i\}$$

- But then

$$\boldsymbol{r}_{i+1} \perp \text{span}\{\boldsymbol{v}_0 \ldots \boldsymbol{v}_i\} \Rightarrow \boldsymbol{r}_{i+1} \perp \text{span}\{\boldsymbol{r}_0 \ldots \boldsymbol{r}_i\} \Rightarrow \boldsymbol{r}_{i+1} \cdot \boldsymbol{r}_j = 0 \;\; \forall j \leq i$$

*we could actually show that $\text{span}\{\boldsymbol{r}_0 \ldots \boldsymbol{r}_i\} = K_i(\boldsymbol{A}, \boldsymbol{r}_0)$

Conjugate Gradient Magic

Now the CG magic, is that taking into account that $\boldsymbol{r}_{i+1} \cdot \boldsymbol{r}_j = 0\ \forall j \leq i$ we discover that **many of the β_* are actually 0** … hence **no need to store the search vectors**.

Proof: $\boldsymbol{r}_{i+1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{i+1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_i - \boldsymbol{\alpha}\boldsymbol{A}\boldsymbol{v}_i = \boldsymbol{r}_i + \alpha_i \boldsymbol{A}\boldsymbol{v}_i \Rightarrow \alpha_i \boldsymbol{A}\boldsymbol{v}_i = \boldsymbol{r}_{i+1} - \boldsymbol{r}_i$

Hence

$$\boldsymbol{\beta}_{(i+1)k} = -\frac{\boldsymbol{v}_k^t \boldsymbol{A}\boldsymbol{r}_{i+1}}{\boldsymbol{v}_k^t \boldsymbol{A}\boldsymbol{v}_k} = \frac{\boldsymbol{r}_{i+1}^t \boldsymbol{A}\boldsymbol{v}_k}{\boldsymbol{v}_k^t \boldsymbol{A}\boldsymbol{v}_k} \forall \boldsymbol{k} \leq \boldsymbol{i} \quad \rightarrow \quad \boldsymbol{\beta}_{(i+1)k} = \frac{\boldsymbol{r}_{i+1}^t \boldsymbol{r}_{k+1} - \boldsymbol{r}_{i+1}^t \boldsymbol{r}_k}{\boldsymbol{\alpha}_i \boldsymbol{v}_k^t \boldsymbol{A}\boldsymbol{v}_k} = \frac{-\boldsymbol{r}_{i+1}^t \boldsymbol{r}_{k+1}}{\boldsymbol{\alpha}_i \boldsymbol{v}_k^t \boldsymbol{A}\boldsymbol{v}_k}$$

Now $k < i \Rightarrow\ \boldsymbol{r}_k^t \boldsymbol{r}_i = \boldsymbol{0}$ it follows that the only non zero beta is for $k = i$

$$\boldsymbol{\beta}_{(i+1)i} = \frac{-\boldsymbol{r}_{i+1}^t \boldsymbol{r}_{i+1}}{\boldsymbol{\alpha}_i \boldsymbol{v}_i^t \boldsymbol{A}\boldsymbol{v}_i} = \frac{-\boldsymbol{r}_{i+1}^t \boldsymbol{r}_{i+1}}{\boldsymbol{v}_i^t \boldsymbol{r}_{i+1} - \boldsymbol{v}_i^t \boldsymbol{r}_i} = \frac{\boldsymbol{r}_{i+1}^t \boldsymbol{r}_{i+1}}{\boldsymbol{v}_i^t \boldsymbol{r}_i}$$

The last step is to observe that

$$v_i = r_i + \sum_{k=0}^{i-1} \beta_{ik} v_k \Rightarrow \boldsymbol{r}_i^t \boldsymbol{v}_i = \boldsymbol{r}_i^t \boldsymbol{r}_i + \sum_{\boldsymbol{k}=\boldsymbol{0}}^{\boldsymbol{i}-\boldsymbol{1}} \boldsymbol{\beta}_{ik} \boldsymbol{r}_i^t \boldsymbol{v}_k = \boldsymbol{r}_i^t \boldsymbol{r}_i$$

Which allows to conclude that

$$\boldsymbol{\beta}_{(i+1)i} = \frac{\boldsymbol{r}_{i+1}^t \boldsymbol{r}_{i+1}}{\boldsymbol{r}_i^t \boldsymbol{r}_i} \rightarrow \boldsymbol{v}_{i+1} = \boldsymbol{r}_{i+1} + \frac{\boldsymbol{r}_{i+1}^t \boldsymbol{r}_{i+1}}{\boldsymbol{r}_i^t \boldsymbol{r}_i} \boldsymbol{v}_i$$

CONJUGATE GRADIENT ALGORITHM

1. $\boldsymbol{r_0 = b - Ax_o \rightarrow v_0 = r_0}$
2. $\boldsymbol{\alpha_i = \frac{v_i^t r_i}{v_i^t A v_i} \rightarrow x_{i+1} = x_i + \alpha_i v_i}$
3. $\boldsymbol{r_{i+1} = r_i - \alpha_i A v_i}$ could also do $\boldsymbol{r_{i+1} = r_i - Ax_{i+1}}$ but that would require one more matrix-vector product
4. $\boldsymbol{\beta_{(i+1)i} = \frac{r_{i+1}^t r_{i+1}}{r_i^t r_i} \rightarrow v_{i+1} = r_{i+1} + \beta_{(i+1)i} v_i}$
5. *Go back to 2 and loop until convergence*

COST: 1 product $\boldsymbol{Av_i}$ + a few inner products. Guaranteed to converge in N iterations, but will most likely converge much before

Convergence Estimates

Considering the condition number $k = k(A) = {}^{\lambda_{max}}/_{\lambda_{min}}$

Convergence estimate of **CG** is:

$$||e\_i|| < 2\left(\frac{\sqrt{k}-1}{\sqrt{k}+1}\right)^i ||e\_0||$$

Convergence estimate of **Steepest Descent** was:

$$||e\_i|| < \left(\frac{k-1}{k+1}\right)^i ||e\_0||$$

What if the matrix is not symmetric?

CG can only be applied if the matrix is SPD, however for an arbitrary matrix $\boldsymbol{A}$ (non SPD, and eventually not square) one may solve

$$\boldsymbol{A^t A x = A^t b}$$

Variation of the CG algorithm exist in which $\boldsymbol{A^t A}$ is never computed explicitly (good, since $\boldsymbol{A^t A}$ has more nonzeros than $\boldsymbol{A}$)

PROBLEM: the condition number of $\boldsymbol{k(A^t A) = k(A)^2}$ hence the convergence is much slower.

GMRES ALGORITHM

The most known work horse for solving non-SPD systems is the GMRES algorithm. Although we will not go in detail, the iterate of the gmres is

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_0 + \boldsymbol{V}\boldsymbol{y} \qquad \boldsymbol{V} := \begin{pmatrix} \boldsymbol{v}_0 & \cdots & \boldsymbol{v}_i \\ \downarrow & & \downarrow \end{pmatrix}$$

Where $\boldsymbol{y}$ is chosen so to minimize

$$\|\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{V}\boldsymbol{y}\|_2$$

The crucial difference with CG is that since A is not symmetric (nor positive definite) **we need to store all of the $\boldsymbol{v}_i$**

The crucial issue, aside of the **memory occupation**, is **how to effectively perform the minimization** of $\|\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{V}\boldsymbol{y}\|_2$

NOTE: Implementing GMRES i way more technical than CG. USE LIBRARIES!

Using Krylov methods as Matrix-Free

A very interesting property of laplacian methods is that the actual knowledge of the matrix entries $\boldsymbol{A}_{ij}$ is not needed.

One only needs to evaluate the "action of a matrix onto a vector", that is, how to compute $\boldsymbol{A}\boldsymbol{v}$ for any given vector $\boldsymbol{v}$.

A practical example helps in understanding this better:

$$\boldsymbol{A} \coloneqq \begin{pmatrix} \mathbf{2} & \mathbf{-1} & \mathbf{0} \\ \mathbf{-1} & \mathbf{2} & \mathbf{-1} \\ \mathbf{0} & \mathbf{-1} & \mathbf{2} \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \end{pmatrix} \rightarrow \mathbf{Ax} = \begin{pmatrix} \mathbf{2}\boldsymbol{x}_1 - \boldsymbol{x}_2 \\ -\boldsymbol{x}_1 + \mathbf{2}\boldsymbol{x}_2 - \boldsymbol{x}_3 \\ -\boldsymbol{x}_2 + \mathbf{2}\boldsymbol{x}_3 \end{pmatrix}$$

For the CG it would be sufficient to know that the function

$$\mathbf{f}(\mathbf{x}) := \begin{pmatrix} \mathbf{2}\boldsymbol{x}_1 - \boldsymbol{x}_2 \\ -\boldsymbol{x}_1 + \mathbf{2}\boldsymbol{x}_2 - \boldsymbol{x}_3 \\ -\boldsymbol{x}_2 + \mathbf{2}\boldsymbol{x}_3 \end{pmatrix}$$

Can be called whenever A@x is needed

TODOS:

1. Implement a CG for the laplacian problems
2. In the test implementation, verify that the expected orthogonality conditions are met (use “large” laplacian matrices)
3. Implement a matrix free solution using scipy’s algorithm
4. Use the matrix-free approach to impose that the solution of a Laplacian problem **without dirichlet conditions** is zero on average in the domain.

References:

The bible of iterative methods:

https://www-users.cs.umn.edu/~saad/IterMethBook_2ndEd.pdf

An in depth dive into the CG

https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf