Krylov methods

1 Conjugate-gradient algorithm

1.1 Steepest gradient descent and conjugate-gradient algorithms

In this exercise, we compare the steepest gradient descent with the conjugate-gradient algorithm.

Algorithm 1 Steepest descent gradient

function STEEPESTDESCENT($A, b, \varepsilon_{\text{tol}}$)
  $x = 0$
  $p = b$
  while $\|p\| > \varepsilon_{\text{tol}}$ **do**
    $\alpha = \frac{\|p\|^2}{\langle p, Ap \rangle}$
    $x = x + \alpha p$
    $p = p - \alpha Ap$
  end while
  return x
end function

1. Implement the steepest gradient descent algorithm.

2. Implement the conjugate-gradient algorithm.

3. Test on $Tx_* = b$ where $T \in \mathbb{R}^{n \times n}$ is the one-dimensional discrete Laplacian (*i.e.* the tridiagonal matrix with $T_{ii} = 2$ for $1 \leq i \leq n$ and $T_{i,i-1} = T_{i-1,i} = -1$ for $2 \leq i \leq n$), and b is a random vector. Compare the speed of convergence of the steepest gradient descent and the conjugate gradient for $n = 1000$.

4. Plot the error $\|x^{(k)} - x_*\|_T = \sqrt{\langle x^{(k)} - x_*, T(x^{(k)} - x_*) \rangle}$ for $n = 1000$ and x_* computed using the \ command in `LinearAlgebra`. Compare this error with $2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k$ with $\kappa = \frac{\lambda_n}{\lambda_1}$.

Algorithm 2 Conjugate-gradient algorithm

```
function CG(A, b, x^(0), ε_tol)
    p_0 = r^(0) = b − A x^(0), k = 0
    while ‖r^(k)‖ > ε_tol do
        k = k + 1
        α_{k−1} = ‖r^(k−1)‖^2 / ⟨p_{k−1} A p_{k−1}⟩
        x^(k) = x^(k−1) + α_{k−1} p_{k−1}
        r^(k) = r^(k−1) − α_{k−1} A p_{k−1}
        ω_k = ‖r_k‖^2 / ‖r_{k−1}‖^2
        p_k = r^(k) + ω_k p_{k−1}
    end while
    return x^(k)
end function
```

5. Add on the previous plot the curves $2\left(\frac{\sqrt{\kappa_\ell}-1}{\sqrt{\kappa_\ell}+1}\right)^k$ with $\kappa_\ell = \frac{\lambda_{n-\ell}}{\lambda_1}$ for different choices of ℓ.

1.2 Preconditioned conjugate gradient algorithm

1. Implement the preconditioned conjugate gradient algorithm.

We will test the preconditioned conjugate gradient algorithm on the following problem

$$\begin{cases} -\Delta u + \big(x - \dfrac{1}{2}\big)^2 u = f \\ u(0) = u(1) = 0. \end{cases}$$

We use a Fourier discretisation for the solution, *i.e.* we write the solution as $u(x) = \sum_{k=1}^{\infty} u_k \sin(\pi k x)$, and we truncate the corresponding series to a level $k \leq N$.

If f is sufficiently regular (for example smooth), we know that the truncation $u_N(x) = \sum_{k=1}^{N} u_k \sin(\pi k x)$ is close to the solution u in the sense that $\|u_N - u\|_{L^2} = o(N^{-\alpha})$ for any $\alpha > 0$ (*i.e.* the convergence is faster than any inverse polynomial).

To obtain an approximation of u_N, we solve the following linear system

$$D_N v + P_N v = f_N,$$

where

- $D_N \in \mathbb{R}^{N\times N}$ is the diagonal matrix $D_N = \mathrm{diag}(\pi^2, 2^2\pi^2, \dots, N^2\pi^2)$
- $P_N \in \mathbb{R}^{N\times N}$ is the matrix such that for $1 \leq k, \ell \leq N$

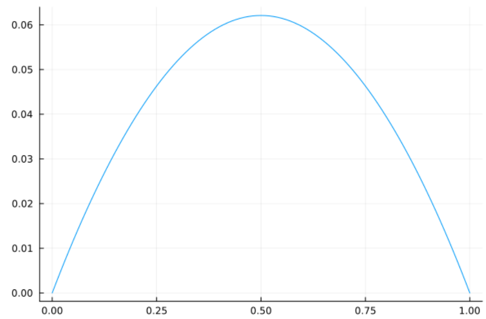
$$(P_N)_{k\ell} = \int_0^1 \sin(\pi k x)\sin(\pi \ell x)\big(x - \frac{1}{2}\big)^2 \,\mathrm{d}x = \begin{cases} \dfrac{1}{12} & \text{if } k = \ell \\ \dfrac{(-1)^{k-\ell}+1}{2\pi^2(k-\ell)^2} - \dfrac{(-1)^{k+\ell}+1}{2\pi^2(k+\ell)^2} & \text{else.} \end{cases}$$

- $f_N \in \mathbb{R}^N$ is the vector $(f_N)_k = \int_0^1 f(x)\sin(k\pi x)\,\mathrm{d}x = \frac{1-(-1)^k}{\pi k}$, for $1 \le k \le N$.

1. Implement the matrices D_N, P_N and f_N.

2. Solve the linear system for $N = [100, 200, 500, 1000, 2000]$ using the conjugate-gradient algorithm with $\varepsilon_{\text{tol}} = 10^{-6}$.

3. Write a function `value(v,x)` that takes a vector $v \in \mathbb{R}^N$ and a scalar x and that returns $\sum_{k=1}^N v_k \sin(\pi k x)$. Check that you obtain a similar plot than below



4. Plot the number of iterations of the conjugate-gradient algorithm to reach this accuracy.

5. Use the preconditioned conjugate-gradient algorithm with $M = D_N$ and plot the number of iterations for the same values of N as in the previous question and with $\varepsilon_{\text{tol}} = 10^{-6}$.

1.3 Inexact iterations

Iterative methods only require to define the matrix vector product Av instead of having to assemble the full matrix A.

In numerous applications, it is sometimes too costly to have the matrix vector product Av up to machine precision. In these cases, instead of computing Av, we have $Av + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon}$ is the error when computing Av.

The academic example that will be explored here is the discretisation using finite-differences of the problem

$$\begin{cases} (-\Delta + x^2 + (-\Delta + 1)^{-1})u = f, & \text{in } [-L, L] \\ u(L) = u(-L) = 0. \end{cases}$$

In the following, we will pick $L = 5$ and $f(x) = \exp(-x^2)$.

The discretised equation is then

$$(-\Delta_N + V_N + (-\Delta_N + I_N)^{-1})u_N = f_N, \tag{1}$$

where

- $-\Delta_N$ is the tridiagonal matrix $(-\Delta_N)_{ii} = \frac{(N+1)^2}{2L^2}$ for $1 \leq i \leq N$ and $(-\Delta_N)_{i,i+1} = \frac{(N+1)^2}{4L^2}$ for $1 \leq i \leq N-1$;
- V_N is the diagonal matrix with entries $((-L+h)^2, (-L+2h)^2, \dots, (L-h)^2)$ with $h = \frac{2L}{N+1}$;
- f_N is the diagonal matrix with entries $(\exp(-(-L+h)^2), \exp(-(-L+2h)^2), \dots, \exp(-(L-h)^2))$ with $h = \frac{2L}{N+1}$.

We want to solve Eq. (1) using an iterative method, the issue is that we do not have access to $(-\Delta_N + I)^{-1}$, hence each matrix-vector multiplication with $(-\Delta_N + V_N + (-\Delta_N + I_N)^{-1})$ requires to solve a linear system with $-\Delta_N + I$. This last linear system is solved using also an iterative method.

1. Let $A \in \mathbb{R}^{N\times N}$ be a symmetric positive-definite matrix. Consider the Richardson iteration with fixed step size $\alpha > 0$: for all $k \geq 0$

$$\begin{cases} r^{(k+1)} = r^{(k)} - \alpha A r^{(k)} \\ x^{(k+1)} = x^{(k)} + \alpha r^{(k)}, \end{cases}$$

   where $x^{(0)} = 0$ is some vector and $r^{(0)} = b$. Show that $(x^{(k)})$ converges to $x_* = A^{-1}b$ for any b if and only if $0 < \alpha < \frac{1}{\lambda_{max}}$ with λ_{max} the largest eigenvalue of A.

2. Consider the *inexact* Richardson iteration with fixed step size $\alpha > 0$: for all $k \geq 0$

$$\begin{cases} \tilde{r}^{(k+1)} = \tilde{r}^{(k)} - \alpha(A\tilde{r}^{(k)} + \boldsymbol{\varepsilon}^{(k)}) \\ x^{(k+1)} = x^{(k)} + \alpha \tilde{r}^{(k)}, \end{cases}$$

   where $x^{(0)} = 0$ and $\tilde{r}^{(0)} = b$.

   (a) Show that for any $k \geq 0$, $\tilde{r}^{(k)} - r^{(k)} = \alpha \sum_{j=1}^{k} (I - \alpha A)^{k-j} \boldsymbol{\varepsilon}^{(j-1)}$.

   (b) Deduce that if for $j \geq 0$, $\|\boldsymbol{\varepsilon}^{(j)}\| \leq \tau \|A\|$, then $\|\tilde{r}^{(k)} - r^{(k)}\| \leq \tau\, \mathrm{cond}_2(A)$.

   (c) Suppose that A is well-conditioned. What can be said on the speed of convergence of the exact and inexact methods?

3. We now go back to (1) and solve this equation using a conjugate-gradient method, up to accuracy τ on the residual. We want to check that a fairly large tolerance τ_{in} can be selected for solving approximately $v^{(k)} = (-\Delta_N + I)^{-1} u^{(k)}$, where $u^{(k)}$ is the k-th iteration of the conjugate-gradient algorithm. Write the algorithm for solving (1) with the conjugate-gradient algorithm with tolerance τ, where the inner linear system $v^{(k)} = (-\Delta_N + I)^{-1} u^{(k)}$ is solved with a conjugate-gradient with tolerance τ_{in}.

4. Perform tests for $N = 1000$ and $\tau_{in} = 10^k \tau$, for $k = -3, \dots, 1$, and compute the exact residual at the end the inexact conjugate-gradient algorithm.

2 GMRES

2.1 GMRES and restarted GMRES

We recall the algorithms that will be needed here.

Algorithm 3 Arnoldi algorithm

function ARNOLDI(A, v, k)
  `V=zeros(n,k+1)`
  `H=zeros(k+1,k)`
  $V[:,1] = \frac{v}{\|v\|}$
  for $j = 1, \dots, k$ **do**
    for $i = 1, \dots, j$ **do**
      $H[i,j] = \langle V[:,i], AV[:,j] \rangle$[1]
    end for
    $\widehat{v}_{j+1} = AV[:,j] - \sum_{i=1}^{j} H[i,j]V[:,i]$
    $H[j+1,j] = \|\widehat{v}_{j+1}\|$
    if $h_{j+1,j} \neq 0$ **then**
      $V[:,j+1] = \frac{\widehat{v}_{j+1}}{H[j+1,j]}$
    end if
  end for
  return `V`,`UpperHessenberg(H)`
end function

We recall that in exact arithmetics, we have $AV[:,1:k] = VH$.

For `H` of type `UpperHessenberg`, some linear algebra methods are more efficient exploiting the upper Hessenberg structure of the matrix. In particular, for `H` of size $m \times n$ with $m > n$, `q,r=qr(H)` returns `q` which is a compact representation of an orthogonal matrix of size $m \times m$ and `r` an upper triangular matrix of size $n \times n$.

Algorithm 4 GMRES

function GMRES($A, b, x^{(0)}, K$)
  $r^{(0)} = b - Ax^{(0)}$, $k = 0$
  $V, H =$ `arnoldi(`$A, r^{(0)}, K$`)`
  `q,r=qr(H)`
  $\begin{bmatrix} g_K \\ \gamma_{K+1} \end{bmatrix} = \|r^{(0)}\| Q^T e_1$ ▷ e_1 1st canonical vector of $\mathbb{R}^{K+1}$
  `t=r`$^{-1} g_K$
  $\|r^{(K)}\| = |\gamma_{K+1}|$
  return $x^{(0)} + V[:,1:K]t, \|r^{(K)}\|$
end function

1. Implement the Arnoldi algorithm `arnoldi(A,v,k)` that returns `V,H` where `V` is a `Matrix` and `H` is `UpperHessenberg`.

2. Implement the version of the GMRES algorithm described above.

3. Test the GMRES algorithm for the linear system $T_{\alpha,\sigma}x = b$, where $T_{\alpha,\sigma} \in \mathbb{R}^{n\times n}$ is the tridiagonal matrix $\mathrm{tridiag}(-\alpha, \sigma+\alpha, -\frac{1}{\alpha})$, for $n = 50$ and $\sigma = \alpha = 2$.

4. For $n = 200$, $\sigma = 2$, $\alpha = 0.9$, plot the convergence of the residuals as a function of K.

5. For $n = 200$, $\sigma = 1.1$, $\alpha = 0.9$, plot the convergence of the residuals as a function of K.

6. Implement a restarted GMRES `rgmres(A,b,x0,K,nb_restart)`, where `K` is the number of inner GMRES iterations and `nb_restart` the number of restarts.

7. Test it on the previous examples and plot the behaviour of the convergence with respect to the restart parameter K.

2.2 Matrix-free problem

Let $A, B \in \mathbb{R}^{N\times N}$, $C \in \mathbb{R}^{N\times N}$. We consider the following equation on $X \in \mathbb{R}^{N\times N}$

$$AX + XB = C. \tag{2}$$

The left-hand side is a linear operator $\mathcal{L}$ acting on matrices $\mathbb{R}^{N\times N}$, where $\mathcal{L}(X) = AX+XB$ for $X \in \mathbb{R}^{N\times N}$.

We want to solve this matrix equation using GMRES.

1. Implement `matvec(A,X)` which returns the matrix $\mathcal{L}(X) = AX + XB$.

2. Adapt the function `gmres` such that it only requires the linear operator $\mathcal{L}$ and not the full matrix.

3. Solve the equation (2) with $A, B \in \mathbb{R}^{N\times N}$ are of form $\frac{1}{\sqrt{N}}$`randn(N,N)`$+I_N$ and $C \in \mathbb{R}^{N\times N}$ is a random matrix with the restarted GMRES algorithm with different restart parameters.