Test of iterativemethods - Trefethen’sexample

Taken from Trefethen, Bau: Numerical Linear Algebra, SIAM 1997, p. 315

Preliminaries

Clear all symbols from previous evaluations to avoid conflicts

```
Clear["Global`*"]
```

Problem

Solve $Ax = b$ for a very sparse array

```
In[1]:= n = 1000;
        A = SparseArray[{{i_, i_} → 0.5 + Sqrt[i],
            {i_, j_} /; Abs[i - j] == 1 → 1.0, {i_, j_} /; Abs[i - j] == 100 → 1.0}, {n, n}]
        b = ConstantArray[1, n];
        xDirect = LinearSolve[A, b];

Out[2]= SparseArray[ Specified elements: 4798
                     Dimensions: {1000, 1000} ]
```

```
In[5]:= Norm[A.xDirect - b]
```

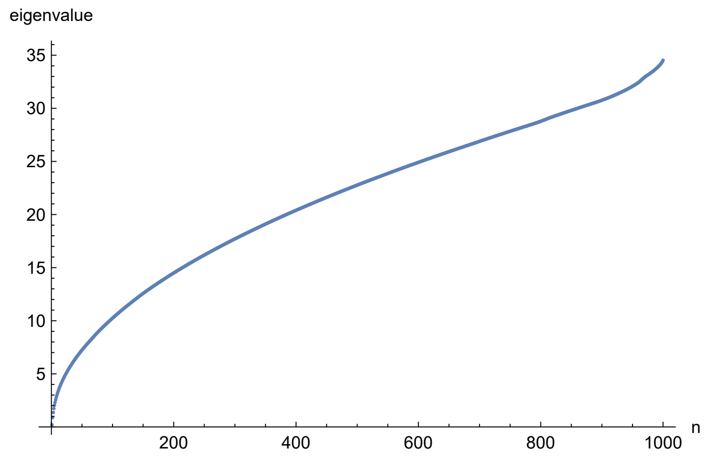
Out[5]= $5.54111410687 \times 10^{-15}$

The matrix is symmetric positive definite, all eigenvalues are positive. Thus all methods should converge.

```
In[62]:= eigenA = Sort[Eigenvalues[A]];
         Print["ϰ(A)  = ", Max[eigenA] / Min[eigenA]]
         ListPlot[{eigenA}, PlotRange → All, AxesLabel → {"n", "eigenvalue"}]
```

```
••• Eigenvalues: Because finding 1000 out of the 1000 eigenvalues and/or eigenvectors is likely to be faster with dense matrix
      methods, the sparse input matrix will be converted. If fewer eigenvalues and/or eigenvectors would be sufficient, consider
      restricting this number using the second argument to Eigenvalues.

ϰ(A)  = 173.448839396
```

Out[64]=



Basic iterative methods

In general, we solve a system of linear equations

$$A\,x = b \tag{1}$$

and use a standard decomposition

$$A = D - \tilde{L} - \tilde{U} = D(I - L - U) \tag{2}$$

where D is the diagonal of A,

$-\tilde{L}$ and $-\tilde{U}$ are the strictly lower and upper triangular parts respectively and $L = D^{-1}\tilde{L}$, $U = D^{-1}\tilde{U}$,

More generally will write $A = M - K$.

One step of the general iterative method can be written as

$$x_{m+1} = R\,x_m + c \text{ where } R = M^{-1}K,\ c = M^{-1}b \tag{3}$$

and the method is convergent if and only if the spectral radius of the matrix R

$$\rho(R) = \max |\lambda_i| \quad \text{where } \lambda_i \text{ are eigenvalues of } R \tag{4}$$

satisfies

$$\rho(R) < 1. \tag{5}$$

```
In[12]:= Diag = DiagonalMatrix[Diagonal[A]];
         D1 = Inverse[Diag];
         L = -D1.LowerTriangularize[A, -1];
         U = -D1.UpperTriangularize[A, 1];
         Id = IdentityMatrix[n];
         (* MatrixForm[U] *)
```

An initial guess and the maximum number of iterations will be the same for all methods.

```
In[17]:= x0 = ConstantArray[0.0, n];
         niter = 200;
```

And here is a common part of all basic iterative algorithms. It is supposed that n_{iter}, x_0 and x_{direct} are already assigned.

```
In[19]:= basicIterativeMethod[matrixR_, vectorC_] :=
          Module[{x, e},
           x = x0;
           e = ConstantArray[1.0 × 10^-20, niter + 1];
           e[[1]] = Max[Abs[x - xDirect]];
           Do[
            x = matrixR.x + vectorC;
            e[[i + 1]] = Max[Abs[x - xDirect]],
            {i, 1, niter}
           ];
           {x, e}
          ]
```

Jacobi method

One step of the Jacobi method is

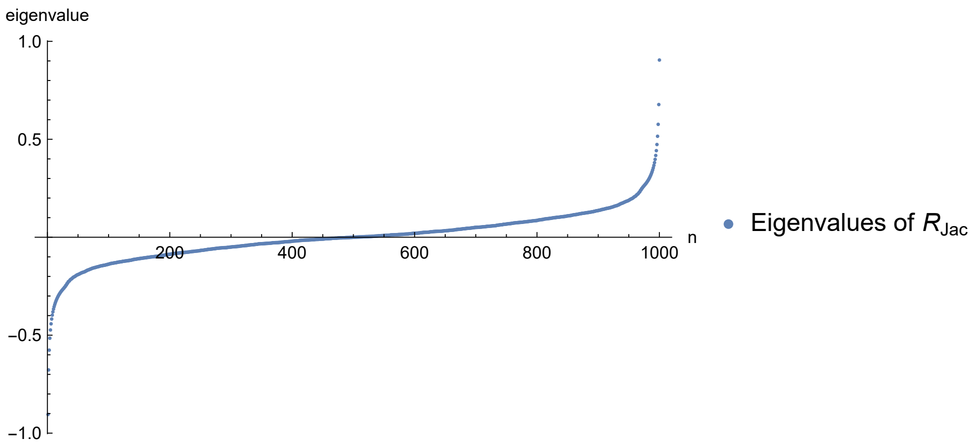
$$x_{m+1,j} = \frac{1}{a_{jj}}\left(b_j - \sum_{k\neq j} a_{jk}\, x_{m,k}\right), j = 1, \;\ldots, n \tag{6}$$

or in a matrix form

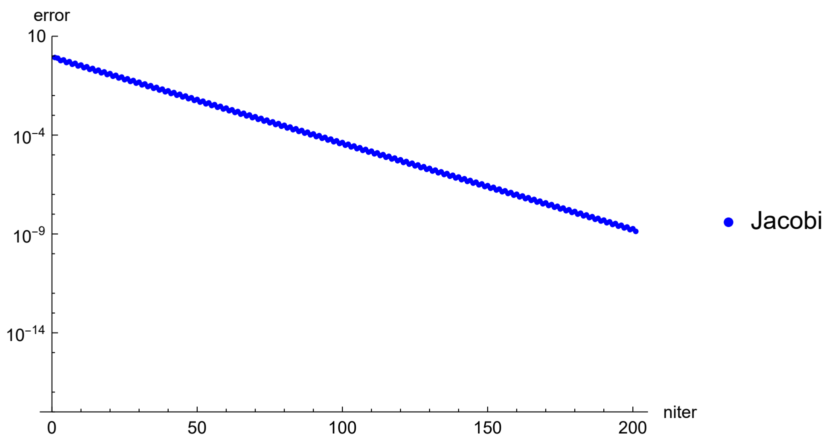
$$x_{m+1} = R_{\text{Jac}}\, x_m + c_{\text{Jac}} \;\; \text{where}\; R_{\text{Jac}} = D^{-1}\left(\tilde{L} + \tilde{U}\right) = L + U \;\text{and}\; c_{\text{Jac}} = D^{-1}\, b \tag{7}$$

```
In[54]:= R = L + U;
         c = D1.b;
         (* MatrixForm[R]*)
         eigen = Eigenvalues[R];
         ρJac = Max[Abs[eigen]];
         Print["ρ(RJac) = ", ρJac]
         ListPlot[Sort[eigen], PlotRange → All,
          PlotLegends → {"Eigenvalues of RJac"}, AxesLabel → {"n", "eigenvalue"}]
         {xIter, errorJac} = basicIterativeMethod[R, c];
         ListLogPlot[{errorJac}, PlotRange → {10^1, 10^(-18)},
          PlotStyle → {Blue}, PlotLegends → {"Jacobi"}, AxesLabel → {"niter", "error"}]

         ρ(RJac) = 0.904511945752
```

Out[59]=



Out[61]=



Gauss-Seidel method

One step of the Gauss-Seidel method is

$$x_{m+1,j} = \frac{1}{a_{jj}}\left(b_j - \sum_{k=1}^{j-1} a_{jk}\, x_{m+1,k} - \sum_{k=j+1}^{n} a_{jk}\, x_{m,k}\right), j = 1, \ldots, n \tag{8}$$

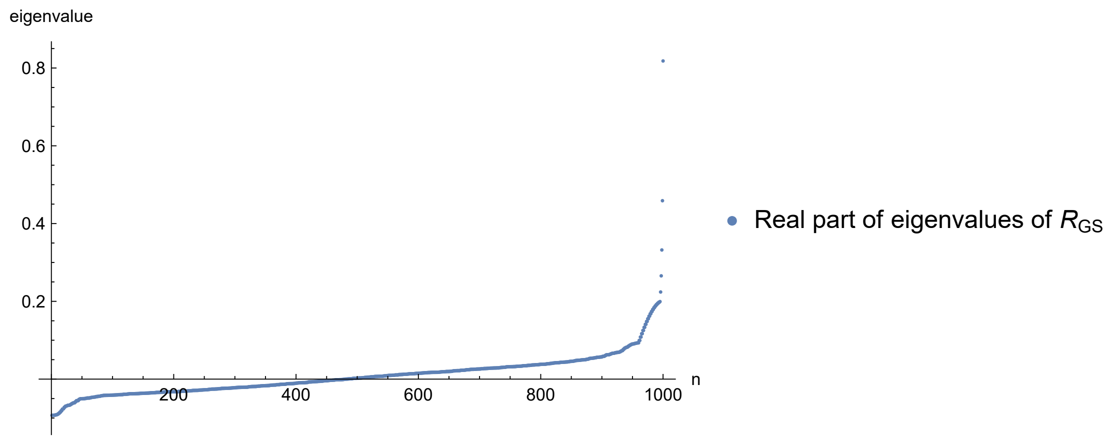
or in a matrix form

$$x_{m+1} = R_{GS}\, x_m + c_{GS} \quad \text{where } R_{GS} = \left(D - \tilde{L}\right)^{-1} \tilde{U} = (I - L)^{-1}\, U \text{ and } c_{GS} = \left(D - \tilde{L}\right)^{-1} b = (I - L)^{-1}\, D^{-1}\, b \tag{9}$$

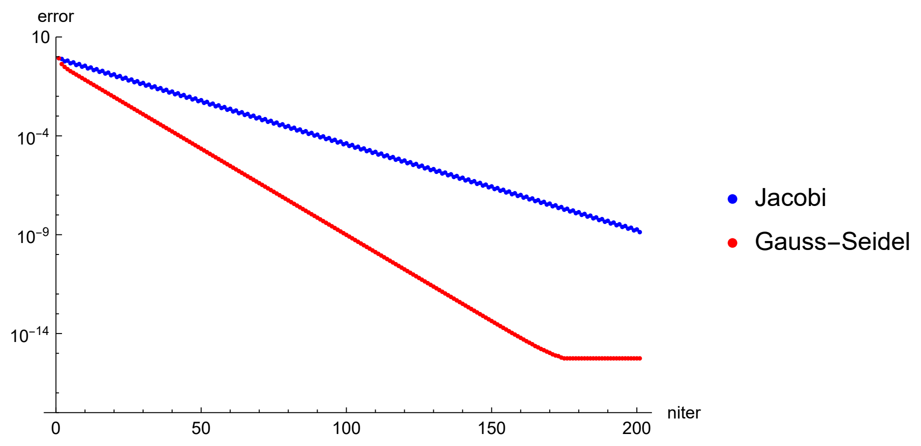
In[45]:=
```
ImL1 = Inverse[Id - L];
R = ImL1.U;
c = ImL1.D1.b;
(*MatrixForm[R]*)
eigen = Eigenvalues[R];
ρGS = Max[Abs[eigen]];
Print["ρ(RGS) = ", ρGS]
ListPlot[Sort[Re[eigen]], PlotRange → All,
 PlotLegends → {"Real part of eigenvalues of RGS"}, AxesLabel → {"n", "eigenvalue"}]
{xIter, errorGS} = basicIterativeMethod[R, c];
ListLogPlot[{errorJac, errorGS}, PlotRange → {10^1, 10^(-18)}, PlotStyle → {Blue, Red},
 PlotLegends → {"Jacobi", "Gauss-Seidel"}, AxesLabel → {"niter", "error"}]
```

```
ρ(RGS) = 0.818141860008
```

Out[51]=



Out[53]=



Successive overrelaxation method with optimal ω

One step of the SOR(ω) method is

$$x_{m+1,j} = (1-\omega)\,x_{m,j} + \frac{\omega}{a_{jj}}\left(b_j - \sum_{k=1}^{j-1} a_{jk}\,x_{m+1,k} - \sum_{k=j+1}^{n} a_{jk}\,x_{m,k}\right), j = 1, \ldots, n \tag{10}$$

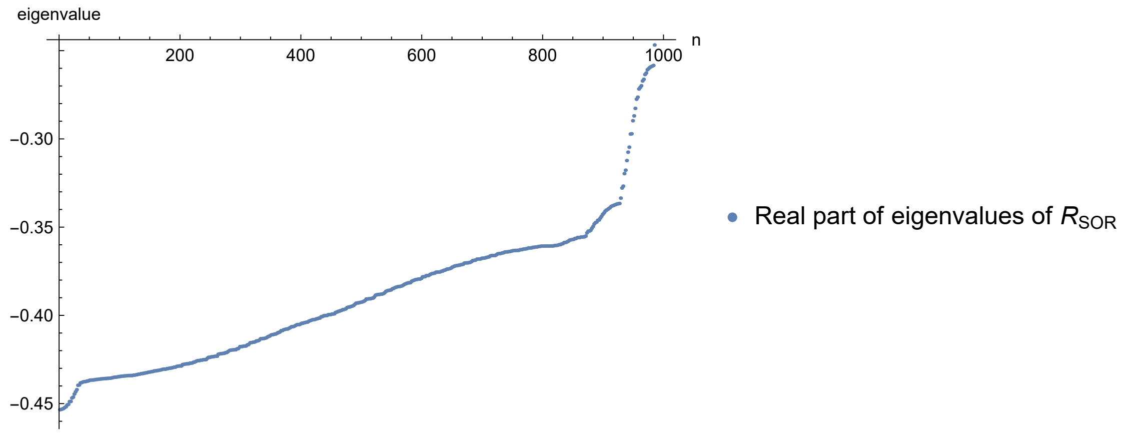
or in a matrix form

$$\begin{array}{l} x_{m+1} = R_{\text{SOR}}\,x_m + c_{\text{SOR}} \;\; \text{where } R_{\text{SOR}} = \left(D - \omega\,\tilde{L}\right)^{-1}\left[(1-\omega)\,D + \omega\,\tilde{U}\right] = (I - \omega\,L)^{-1}[(1-\omega)\,I + \omega\,U] \\ \text{and } c_{\text{GS}} = \omega\left(D - \omega\,\tilde{L}\right)^{-1} b = \omega(I - \omega\,L)^{-1}\,D^{-1}\,b \end{array} \tag{11}$$

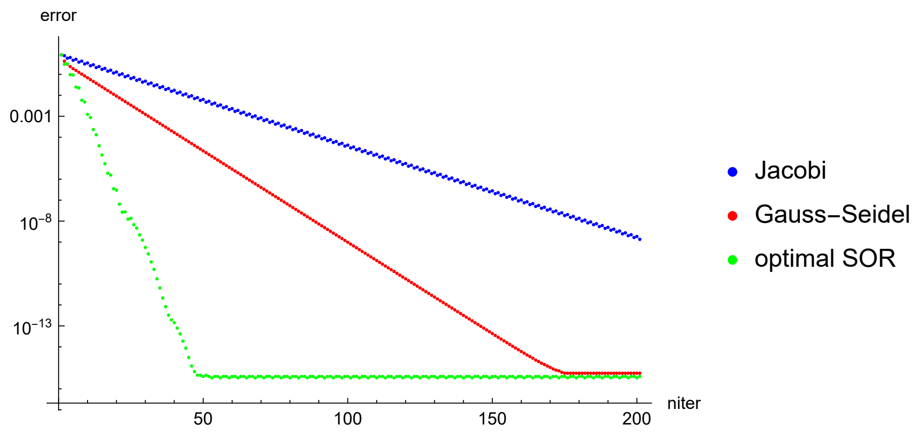
In[76]:=
```
ω = 2 / (1 + Sqrt[1 - ρJac^2]);
Print["ω = ", ω]
ImL1 = Inverse[Id - ω L];
R = ImL1.((1 - ω) Id + ω U);
c = ω ImL1.D1.b;
(*MatrixForm[R]*)
eigen = Eigenvalues[R];
ρSOR = Max[Abs[eigen]];
Print["ρ(R_SOR) = ", ρSOR]
ListPlot[Sort[Re[eigen]],
 PlotLegends → {"Real part of eigenvalues of R_SOR"}, AxesLabel → {"n", "eigenvalue"}]
{xIter, errorSORopt} = basicIterativeMethod[R, c];
ListLogPlot[{errorJac, errorGS, errorSORopt},
 PlotRange → All, PlotStyle → {Blue, Red, Green},
 PlotLegends → {"Jacobi", "Gauss-Seidel", "optimal SOR"}, AxesLabel → {"niter", "error"}]
```

```
ω  =  1.40208377737

ρ(R_SOR)  =  0.47288939388
```

Out[84]=



Out[86]=

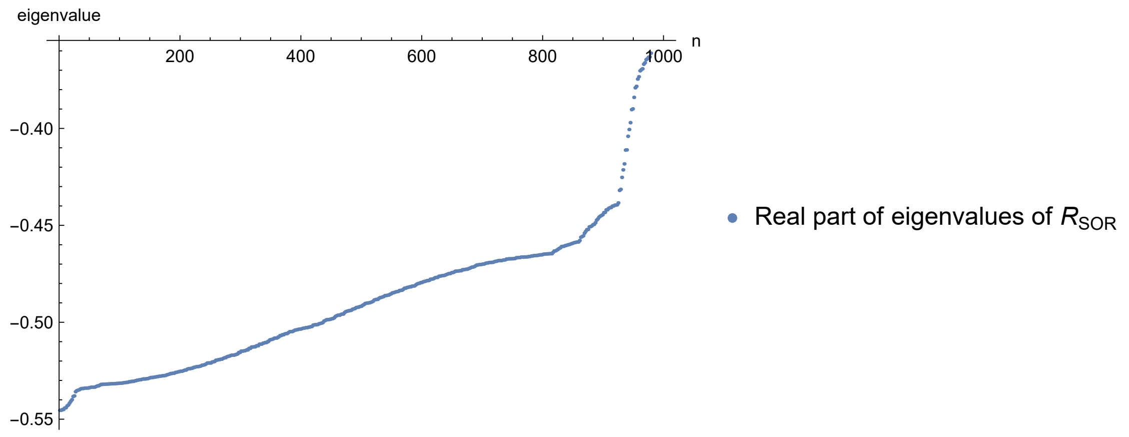


Successive overrelaxation method with estimated ω

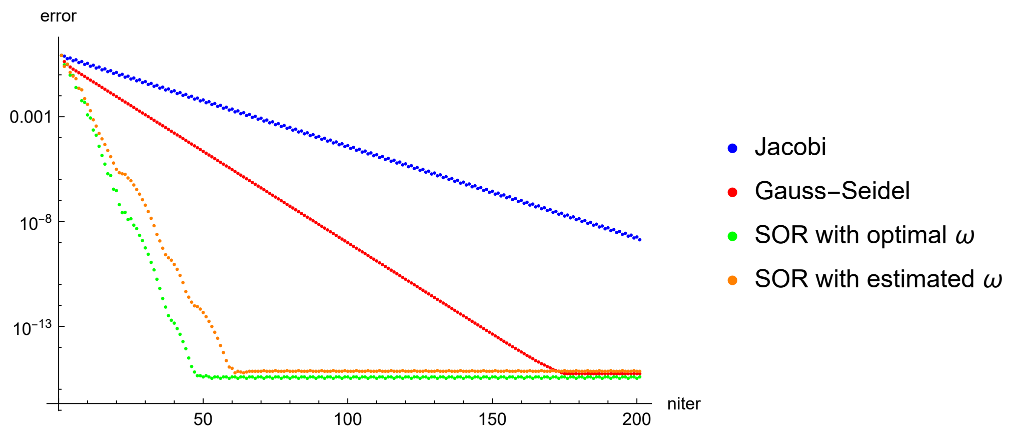
In[87]:=
```
Print["ρ(R_Jac) = ", ρJac]
(* Determination of ρ(R_Jac) from the error *)
nit1 = 5; nit2 = 10;
ρJacEst = Exp[(Log[errorJac[[nit2]]] - Log[errorJac[[nit1]]]) / (nit2 - nit1)];
Print["estimated ρ(R_Jac) = ", ρJacEst]
(* Use this value in SOR algorithm *)
ω = 2 / (1 + Sqrt[1 - ρJacEst^2]);
Print["ω = ", ω]
ImL1 = Inverse[Id - ω L];
R = ImL1.((1 - ω) Id + ω U);
c = ω ImL1.D1.b;
(*MatrixForm[R]*)
eigen = Eigenvalues[R];
ρSORest = Max[Abs[eigen]];
Print["estimated ρ(R_SOR) = ", ρSORest, " (Optimal ρ(R_SOR) = ", ρSOR, ")"]
ListPlot[Sort[Re[eigen]],
 PlotLegends → {"Real part of eigenvalues of R_SOR"}, AxesLabel → {"n", "eigenvalue"}]
{xIter, errorSOR} = basicIterativeMethod[R, c];
ListLogPlot[{errorJac, errorGS, errorSORopt, errorSOR},
 PlotRange → All, PlotStyle → {Blue, Red, Green, Orange},
 PlotLegends → {"Jacobi", "Gauss-Seidel", "SOR with optimal ω", "SOR with estimated ω"},
 AxesLabel → {"niter", "error"}]
```

```
ρ(R_Jac) = 0.904511945752

estimated ρ(R_Jac) = 0.943764794391

ω = 1.50306140916

estimated ρ(R_SOR) = 0.564749817752 (Optimal ρ(R_SOR) = 0.47288939388)
```

Out[99]=



Out[101]=



Gradient iterative methods

Solving a system of linear equations

$$A x = b \tag{12}$$

with a symmetric positive definite matrix *A* by searching a minimum of the function

$$\phi(x) = \frac{1}{2} x^T A x - x^T b \tag{13}$$

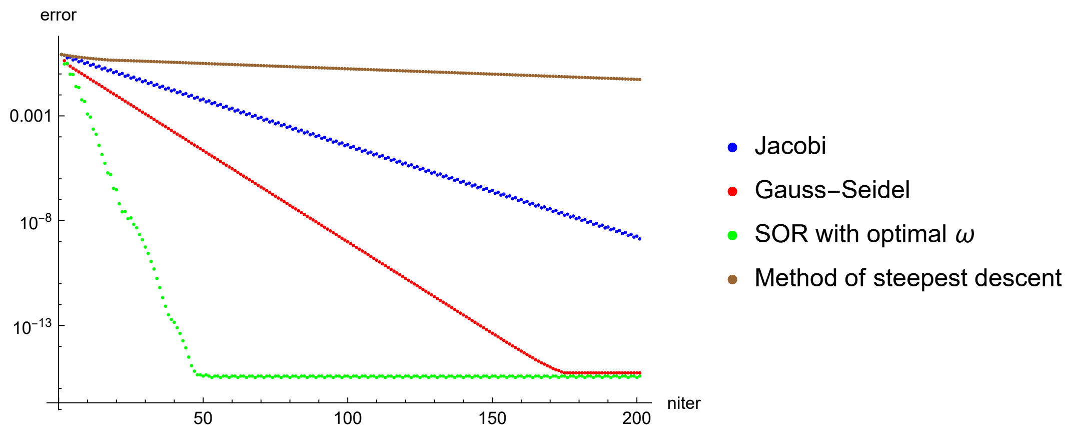
An initial guess and the maximum number of iterations will be the same as in basic iterative method.

Method of the steepest descent

In[102]:=

```
xIter = x0;
r = b - A.x0;
error = ConstantArray[1.0 × 10^-50, niter + 1];
error⟦1⟧ = Max[Abs[xIter - xDirect]];
Do[
 w = A.r;
 α = r.r / r.w;
 xIter = xIter + α r;
 r = r - α w;
 error⟦i + 1⟧ = Max[Abs[xIter - xDirect]],
 {i, 1, niter}
]
errorMSD = error;
ListLogPlot[{errorJac, errorGS, errorSORopt, errorMSD},
 PlotRange → All, PlotStyle → {Blue, Red, Green, Brown},
 PlotLegends → {"Jacobi", "Gauss-Seidel", "SOR with optimal ω",
   "Method of steepest descent"}, AxesLabel → {"niter", "error"}]
```

Out[108]=

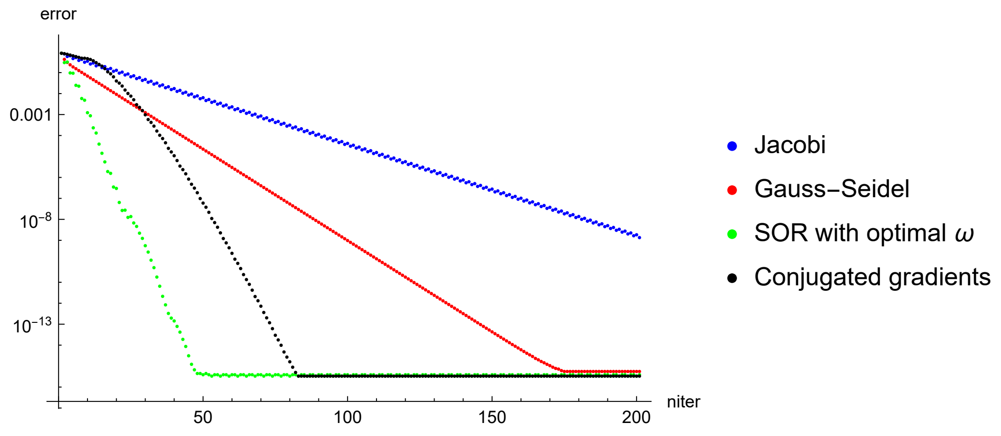


Conjugate Gradient Method

In[109]:=

```
xIter = x0;
r = b - A.x0;
p = r;
γ = r.r;
error = ConstantArray[1.0 × 10^-50, niter + 1];
error[[1]] = Max[Abs[xIter - xDirect]];
Do[
 w = A.p;
 α = γ / p.w;
 xIter = xIter + α p;
 r = r - α w;
 β = 1 / γ;
 γ = r.r;
 β = β γ;
 p = r + β p;
 error[[i + 1]] = Max[Abs[xIter - xDirect]],
 {i, 1, niter}
]
errorCG = error;
ListLogPlot[{errorJac, errorGS, errorSORopt, errorCG},
 PlotRange → All, PlotStyle → {Blue, Red, Green, Black},
 PlotLegends → {"Jacobi", "Gauss-Seidel", "SOR with optimal ω", "Conjugated gradients"},
 AxesLabel → {"niter", "error"}]
```

Out[117]=



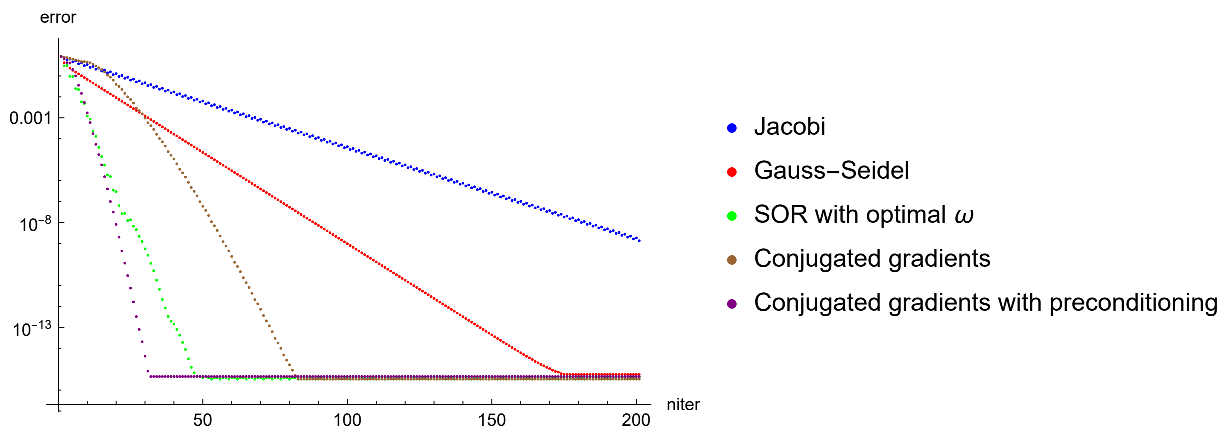
Conjugate Gradient Method with Preconditioning

As a preconditioner we use a diagonal of A

In[118]:=

```
M = DiagonalMatrix[Diagonal[A]];
xIter = x0;
r = b - A.x0;
error = ConstantArray[1.0 × 10^-50, niter + 1];
error⟦1⟧ = Max[Abs[xIter - xDirect]];
Do[
 z = LinearSolve[M, r];
 If[i == 1,
  p = z; γ = r.z,
  β = 1 / γ; γ = r.z; β = β γ; p = z + β p
 ];
 w = A.p;
 α = γ / p.w;
 xIter = xIter + α p;
 r = r - α w;
 error⟦i + 1⟧ = Max[Abs[xIter - xDirect]],
 {i, 1, niter}
]
(* Save results for later comparison with other methods *)
errorCGP = error;
ListLogPlot[{errorJac, errorGS, errorSORopt, errorCG, errorCGP},
 PlotRange → All, PlotStyle → {Blue, Red, Green, Brown, Purple},
 PlotLegends → {"Jacobi", "Gauss-Seidel", "SOR with optimal ω", "Conjugated gradients",
   "Conjugated gradients with preconditioning"}, AxesLabel → {"niter", "error"}]
```

Out[125]=



The condition number of A was

In[126]:=

```
eigenA = Sort[Eigenvalues[A]];
Print["ϰ(A)  = ", Max[eigenA] / Min[eigenA]]
```

```
Eigenvalues: Because finding 1000 out of the 1000 eigenvalues and/or eigenvectors is likely to be faster with dense matrix
    methods, the sparse input matrix will be converted. If fewer eigenvalues and/or eigenvectors would be sufficient, consider
    restricting this number using the second argument to Eigenvalues.
```

```
ϰ(A)  = 173.448839396
```

The condition number of $M^{-1}A$ is

In[128]:=

```
eigenMA = Sort[Eigenvalues[Inverse[M].A]];
Print["ϰ(M^-1A)  = ", Max[eigenMA] / Min[eigenMA]]
```

```
ϰ(M^-1A)  = 19.9450283153
```

Other Krylov space methods

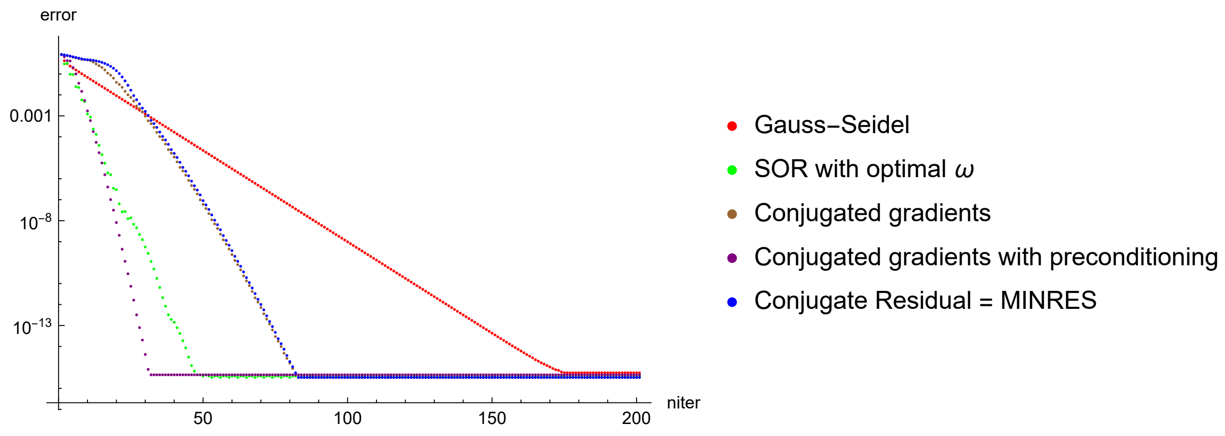
MINRES equivalent to CR algorithm

Special case of GMRES for symmetric, positive definite matrix A called MINRES = minimal residuals. The Arnoldi algorithm is replaced by the Lanczos algorithm (only three-term recurrence) and the whole algorithm produces the same approximation as Stiefel's Conjugate Residual (CR) method which can be written in a very similar way as the CG method (see e.g. report https://web.stanford.edu/group/SOL/reports/SOL-2011-2R.pdf):

In[130]:=

```
xIter = x0;
r = b - A.x0;
s = A.r;
γ = r.s;
p = r;
q = s;
error = ConstantArray[1.0 × 10^-50, niter + 1];
error[[1]] = Max[Abs[xIter - xDirect]];
Do[
 α = γ / (q.q);
 xIter = xIter + α p;
 r = r - α q;
 s = A.r;
 β = 1 / γ;
 γ = r.s;
 β = β γ;
 p = r + β p;
 q = s + β q;
 error[[i + 1]] = Max[Abs[xIter - xDirect]],
 {i, 1, niter}
]
(* Save results for later comparison with other methods *)
errorCR = error;
ListLogPlot[{errorGS, errorSORopt, errorCG, errorCGP, errorCR},
 PlotRange → All, PlotStyle → {Red, Green, Brown, Purple, Blue},
 PlotLegends → {"Gauss-Seidel", "SOR with optimal ω",
   "Conjugated gradients", "Conjugated gradients with preconditioning",
   "Conjugate Residual = MINRES"}, AxesLabel → {"niter", "error"}]
```

Out[140]=



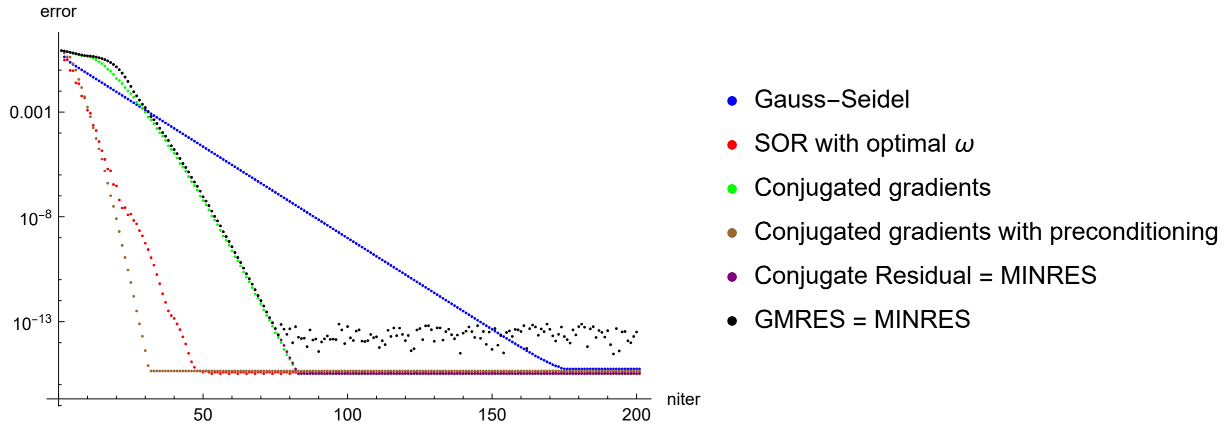
GMRES

This algorithm should produce the same results as CR algorithm if applied to a symmetric, positive definite system.

In[141]:=

```
xIter = x0;
r = b - A.x0;
rr = Norm[r];
error = ConstantArray[1.0 × 10^-50, niter];
error[[1]] = Max[Abs[xIter - xDirect]];
q = ConstantArray[0.0, {n, niter}];
H = ConstantArray[0.0, {niter + 1, niter}];
q[[All, 1]] = r / rr;
Do[
 q[[All, i]] = A.q[[All, i - 1]];
 Do[
  H[[j, i - 1]] = Conjugate[q[[All, j]]].q[[All, i]];
  q[[All, i]] = q[[All, i]] - H[[j, i - 1]] * q[[All, j]],
  {j, 1, i - 1}];
 H[[i, i - 1]] = Norm[q[[All, i]]];
 q[[All, i]] = q[[All, i]] / H[[i, i - 1]];
 y = LeastSquares[H[[1 ;; i, 1 ;; i - 1]], rr * UnitVector[i, 1]];
 xIter = x0 + q[[All, 1 ;; i - 1]].y;
 error[[i]] = Max[Abs[xIter - xDirect]],
 {i, 2, niter}
]
(* Save results for later comparison with other methods *)
errorGMRES = error;
ListLogPlot[{errorGS, errorSORopt, errorCG, errorCGP, errorCR, errorGMRES},
 PlotRange → All, PlotStyle → {Blue, Red, Green, Brown, Purple, Black},
 PlotLegends → {"Gauss-Seidel", "SOR with optimal ω",
   "Conjugated gradients", "Conjugated gradients with preconditioning",
   "Conjugate Residual = MINRES", "GMRES = MINRES"}, AxesLabel → {"niter", "error"}]
```

Out[151]=



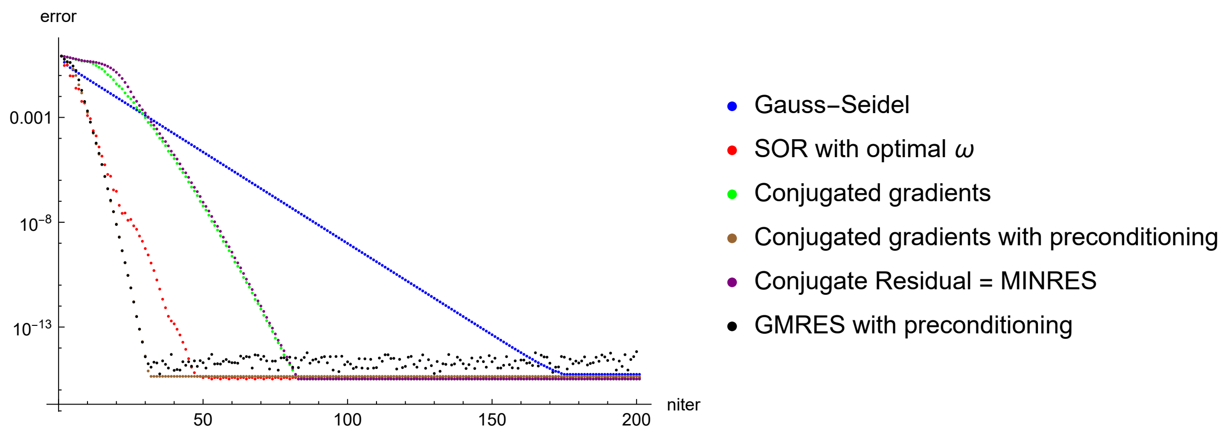
GMRES with preconditioning

Instead of A we use $M^{-1}A$ where M is the diagonal

In[152]:=

```
M = DiagonalMatrix[Diagonal[A]];
MA = Inverse[M].A;
xIter = x0;
r = Inverse[M].b - MA.x0;
rr = Norm[r];
error = ConstantArray[1.0 × 10^-50, niter];
error⟦1⟧ = Max[Abs[xIter - xDirect]];
q = ConstantArray[0.0, {n, niter}];
H = ConstantArray[0.0, {niter + 1, niter}];
q⟦All, 1⟧ = r / rr;
Do[
 q⟦All, i⟧ = MA.q⟦All, i - 1⟧;
 Do[
  H⟦j, i - 1⟧ = Conjugate[q⟦All, j⟧].q⟦All, i⟧;
  q⟦All, i⟧ = q⟦All, i⟧ - H⟦j, i - 1⟧ * q⟦All, j⟧,
  {j, 1, i - 1}];
 H⟦i, i - 1⟧ = Norm[q⟦All, i⟧];
 q⟦All, i⟧ = q⟦All, i⟧ / H⟦i, i - 1⟧;
 y = LeastSquares[H⟦1 ;; i, 1 ;; i - 1⟧, rr * UnitVector[i, 1]];
 xIter = x0 + q⟦All, 1 ;; i - 1⟧.y;
 error⟦i⟧ = Max[Abs[xIter - xDirect]],
 {i, 2, niter}
]
(* Save results for later comparison with other methods *)
errorGMRES = error;
ListLogPlot[{errorGS, errorSORopt, errorCG, errorCGP, errorCR, errorGMRES},
 PlotRange → All, PlotStyle → {Blue, Red, Green, Brown, Purple, Black},
 PlotLegends → {"Gauss-Seidel", "SOR with optimal ω", "Conjugated gradients",
   "Conjugated gradients with preconditioning", "Conjugate Residual = MINRES",
   "GMRES with preconditioning"}, AxesLabel → {"niter", "error"}]
```

Out[164]=


error
10^-8
10^-13
niter
● Gauss–Seidel
● SOR with optimal ω
● Conjugated gradients
● Conjugated gradients with preconditioning
● Conjugate Residual = MINRES
● GMRES with preconditioning