# Conjugate Gradient

June 18, 2019

## 1 Conjugate Gradient Algorithm

[5]:
```
using LinearAlgebra
using Optim
```

Solve

$$\min_x f(x) = \frac{1}{2}x^T Ax + b^T x + a$$

where $A \succ 0$. Setting $\nabla f(x) = 0$, it is equivalent to solve the linear system $Ax = -b$.

[34]:
```
f = x -> 0.5*dot(x,A*x)+dot(b,x)
```

[34]:
```
#9 (generic function with 1 method)
```

### 1.1 A simple example

Adapted from https://www.rose-hulman.edu/~bryan/lottamath/congrad.pdf

Let

$$A = \begin{pmatrix} 3 & 1 & 0 \\ 1 & 2 & 2 \\ 0 & 2 & 4 \end{pmatrix}$$

Consider the function to minimize

$$f(x) = \frac{1}{2}x^T Ax,$$

and assume the we already computed

$$\begin{aligned} d_0 &= (1,0,0) \\ d_1 &= (1,3,0) \\ d_2 &= (2,6,5). \end{aligned}$$

Check that $d_0$, $d_1$ and $d_2$ are $A$-conjugate.

[ ]:
```
A = [ 3.0 1 0 ; 1 2 2 ; 0 2 4]
d0 = [ 1.0 0 0 ]'
d1 = [ 1.0 -3.0 0.0 ]'
d2 = [ -2.0 6.0 -5.0]'

println("$(dot(d0, A*d1)) $(dot(d0, A*d2)) $(dot(d1, A*d2))")
```

[ ]:
```
det(A), eigen(A)
```

Take initial guess $x_0 = (1, 2, 3)$. Compute $x_1$, $x_2$ and $x_3$ using the conjugate gradient algorithm. Is $x_3$ optimal?

$$\nabla f(x) = Ax$$

[ ]:
```
x0 = [1 2 3.0]'
-A*x0
```

[ ]:
```
f(x) = x^T*A*x
```

We have to compute $\alpha_k$, $k = 1, 2, 3$, by solving

$$\min_{\alpha} f(x_k + \alpha d_k)$$

In order to obtain $\alpha_1$, we have to minimize

$$\begin{aligned}
f(x_0 + \alpha d_0) &= \frac{1}{2}\left(\begin{pmatrix}1 & 2 & 3\end{pmatrix} + \alpha\begin{pmatrix}1 & 0 & 0\end{pmatrix}\right)\begin{pmatrix}3 & 1 & 0\\ 1 & 2 & 2\\ 0 & 2 & 4\end{pmatrix}\left(\begin{pmatrix}1\\2\\3\end{pmatrix} + \alpha\begin{pmatrix}1\\0\\0\end{pmatrix}\right)\\
&= \frac{1}{2}\begin{pmatrix}1+\alpha & 2 & 3\end{pmatrix}\begin{pmatrix}3 & 1 & 0\\ 1 & 2 & 2\\ 0 & 2 & 4\end{pmatrix}\begin{pmatrix}1+\alpha\\ 2\\ 3\end{pmatrix}\\
&= \frac{1}{2}\begin{pmatrix}1+\alpha & 2 & 3\end{pmatrix}\begin{pmatrix}5+3\alpha\\ 11+\alpha\\ 16\end{pmatrix}\\
&= \frac{1}{2}((1+\alpha)(5+3\alpha) + 22 + 2\alpha + 48)\\
&= \frac{1}{2}(3\alpha^2 + 8\alpha + 5 + 70 + 2\alpha)\\
&= \frac{3}{2}\alpha^2 + 5\alpha + \frac{75}{2}
\end{aligned}$$

with respect to $\alpha$.

We can obtain it by searching the zero of the derivative with respect to $\alpha$, that is

$$\frac{d}{d\alpha} f(x + \alpha d) = 0$$

or

$$d^T \nabla f(x + \alpha d) = 0$$

Therefore, we must have

$$3\alpha + 5 = 0$$

Thus

$$\alpha_0 = -\frac{5}{3}$$

$$x_1 = x_0 - \frac{5}{3} d_0 = \begin{pmatrix} -\frac{2}{3} \\ 2 \\ 3 \end{pmatrix}$$

We can also directly compute $\alpha_0$ as

$$\alpha_0 = -\frac{d_0^T \nabla f(x_0)}{d_0^T A d_0}$$

[ ]:
```
x0 = [1 ; 2 ; 3.0]
f = A*x0
```

[ ]:
```
d0 = [1 ; 0 ; 0]
0 = -dot(d0,f)/dot(d0,A*d0)
```

[ ]:
```
x1 = x0+0*d0
```

A linesearch from $x_1$ in direction $d_1$ requires us to minimize

$$f(x_1 + \alpha d_1) = \frac{15}{2}\alpha^2 - 28\alpha + \frac{100}{3}$$

which occurs at

$$\alpha_1 = \frac{28}{15},$$

yielding

$$x_2 = x_1 + \frac{28}{15} d_1 = \begin{pmatrix} \frac{6}{5} \\ \frac{-18}{5} \\ 3 \end{pmatrix}.$$

[ ]:
```
1 = -dot(d1,A*x1)/dot(d1,A*d1)
```

[ ]:
```
28/15
```

[ ]:
```
x2 = x1+1*d1
```

The final linesearch from $x_2$ in direction $d_2$ requires us to minimize

$$f(x_2 + \alpha d_2) = 20\alpha^2 - 24\alpha + \frac{36}{5}$$

which occurs at

$$\alpha_2 = \frac{3}{5},$$

yielding

$$x_3 = x_2 + \frac{3}{5} d_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix},$$

which is of course correct.

Similarly, we can compute the new point as

[ ]:
```
2 = -dot(d2,A*x2)/dot(d2,A*d2)
x3 = x2+2*d2
```

## 1.2 A naive implementation

A first version of the conjugate gradient algorithm follows.

[2]:
```
function cg_quadratic(A:: Matrix, b:: Vector, x0:: Vector, trace:: Bool = false)
    n = length(x0)
    x = x0
    g = b+A*x
    d = -g
    if (trace)
        iter = [ x ]
        iterg = [ norm(g) ]
    end
    k = 0

    for k = 1:n-1
        Ad = A*d
        normd = dot(d,Ad)
         = -dot(d,g)/normd
        x += *d
        if (trace)
            iter = [ iter; [x] ]
            iterg = [ iterg; norm(g)]
        end
        g = b+A*x
         = dot(g,Ad)/normd
        d = -g+*d
    end

    normd = dot(d,A*d)
     = -dot(d,g)/normd
    x += *d
    if (trace)
        g = b+A*x # g must be equal to 0
        iter = [ iter; [x] ]
        iterg = [ iterg; norm(g)]
        return x, iter, iterg
    end

    return x
end
```

[2]:
```
cg_quadratic (generic function with 2 methods)
```

Consider the simple example

[29]:
```
A = [2 1; 1 2]
b = [1, 0]
A\(-b)
```

[29]:
```
2-element Array{Float64,1}:
 -0.6666666666666666
  0.3333333333333333
```

Solve

$$\min_{\alpha} f(x) = \frac{1}{2}x^T Ax + b^T x + c$$

Or, equivalently, we solve

$$c + \min_{\alpha} f(x) = \frac{1}{2}x^T Ax + b^T x$$

[30]:
```
cg_quadratic(A, b, [0, 0], true)
```

[30]:
```
([-0.666667, 0.333333], Array{Float64,1}[[0.0, 0.0], [-0.5, 0.0], [-0.666667,
0.333333]], [1.0, 1.0, 0.0])
```

What if $A$ is not positive definite?

[31]:
```
A = [ 1 2 ; 2 1]
A\(-b)
```

[31]:
```
2-element Array{Float64,1}:
  0.33333333333333326
 -0.6666666666666666
```

[32]:
```
cg_quadratic(A, b, [0, 0], true)
```

[32]:
```
([0.333333, -0.666667], Array{Float64,1}[[0.0, 0.0], [-1.0, 0.0], [0.333333,
-0.666667]], [1.0, 1.0, 1.11022e-16])
```

[33]:
```
cg_quadratic(A, b, [1, 1], true)
```

[33]:
```
([0.333333, -0.666667], Array{Float64,1}[[1.0, 1.0], [-0.369863, -0.0273973],
[0.333333, -0.666667]], [5.0, 5.0, 2.22045e-16])
```

[37]:
```
f([1/3,-2/3])
```

[37]:
```
0.16666666666666666
```

[38]:
```
f([0,0])
```

[38]:
```
0.0
```

The conjugate gradient finds the solution of the linear system, and this does correspond to a first-order critical point of the function.

[ ]:
```
f = x -> A*x+b
```

[ ]:
```
x = [1.0/3; -2.0/3]
f(x)
```

[ ]:
```
x = [1; 1]
f(x)
```

[ ]:
```
step= x -> x-*f(x)
```

[ ]:
```
 = 10
dot(step(x),A*step(x))
```

[ ]:
```
, u = eigen(A)
```

[ ]:
```
u
```

[ ]:
```
x = u[:,1]
 = 10
f = x -> 0.5*dot(x,A*x)+dot(b,x)
f(step(x))
```

[ ]:
```
 = 1000
dot(step(x),A*step(x))+dot(b,x)
```

[ ]:
```
f(x)
```

[ ]:
```
x = [1/3.0; -2/3]
f(x)
```

[ ]:
```
cg_quadratic(A, b, x, true)
```

We will need to incorporate a test on $\nabla f(x_k)$!
A more complex example

[ ]:
```
n = 500;
m = 600;
A = randn(n,m);
A = A * A';  # A is now a positive semi-definite matrix
A = A+I # A is positive definite
b = zeros(n)
for i = 1:n
  b[i] = randn()
end
x0 = zeros(n)
```

[ ]:
```
b1 = A\(-b)
```

[ ]:
```
b2, iter, iterg = cg_quadratic(A, b, x0, true);
```

[ ]:
```
norm(b1-b2)
```

[ ]:
```
iterg
```

It works, but do we need to perform all the 500 iterations? We could be satisfied if we are close to the solution. We can mesure the residual of the linear system

$$r = b + Ax$$

that is nothing else the the gradient of the objective function of the quadratic optimization problem.

[ ]:
```
iter
```

We incorporate a convergence test in the function.

[3]:
```
function cg_quadratic_tol(A:: Matrix, b:: Vector, x0:: Vector, trace:: Bool =␣
 ↪false, tol = 1e-8)
    n = length(x0)
    x = x0
```

```
    if (trace)
        iter = [ x ]
    end
    g = b+A*x
    d = -g
    k = 0

    tol2 = tol*tol

     = 0.0

    while ((dot(g,g) > tol2) && (k <= n))
        Ad = A*d
        normd = dot(d,Ad)
         = dot(g,g)/normd
#         = -dot(d,g)/normd
        x += *d
        if (trace)
            iter = [ iter; x ]
        end
        g = b+A*x
         = dot(g,Ad)/normd
        d = -g+*d
        k += 1
    end

    if (trace)
        iter = [ iter; x ]
        return x, iter, k
    end

    return x, k
end
```

[3]: cg_quadratic_tol (generic function with 3 methods)

[4]:
```
x, iter, k = cg_quadratic_tol(A, b, x0, true)
```

```
UndefVarError: A not defined

Stacktrace:

 [1] top-level scope at In[4]:1
```

The number of iterations is

[ ]:
```
k
```

Are we close to the solution?

[ ]:
```
norm(b1-x)
```

[ ]:
```
size(A)
```

which is much less than the problem dimension

### 1.3 Preconditioned conjugate gradient

A basic implementation of a preconditioned conjugate gradient algorithm follows, were $M$ is the inverse of the preconditioner to apply.

[50]:
```
function pcg_quadratic_tol(A:: Matrix, b:: Vector, x0:: Vector, M:: Matrix,
                           trace:: Bool = false, tol = 1e-8)
    n = length(x0)
    x = x0
    if (trace)
        iter = [ x ]
    end
    g = b+A*x
    v = M*g
    d = -v
    k = 0

    tol2 = tol*tol

     = 0.0

    gv = dot(g,v)
    while ((gv > tol2) && (k <= n))
#    while ((dot(g,g) > tol2) && (k <= n))
        Ad = A*d
        normd = dot(d,Ad)
        #gv = dot(g,v)
         = gv/normd
        x += *d
        if (trace)
            iter = [ iter; x ]
        end
        g += *Ad
        v = M*g
        gvold = gv
        gv = dot(g,v)
         = gv/gvold
        d = -v+*d
        k += 1
    end
```

```
    if (trace)
        iter = [ iter; x ]
        return x, iter, k
    end

    return x, k
end
```

[50]: pcg_quadratic_tol (generic function with 3 methods)

Let's check first that when there is no preconditioning, we obtain the same iterates. Set

[13]:
```
M = zeros(n,n)+I
x, iter, k = pcg_quadratic_tol(A, b, x0, M, true)
```

[13]:
```
([-0.0688577, -0.303923, 0.0911491, -0.388753, 0.404543, -0.570684, 0.104285,
-0.220454, 0.0384772, -0.0635818    0.263589, -0.240377, -0.0372104, -0.298747,
0.167712, -0.229557, -0.0135106, -0.400812, 0.171338, -0.0950039], Any[[0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0    0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0], -0.12259, -0.170787, -0.135049, -0.0916016, -0.0251328,
-0.193892, -0.154141, -0.0895074, -0.0547367    0.263589, -0.240377,
-0.0372104, -0.298747, 0.167712, -0.229557, -0.0135106, -0.400812, 0.171338,
-0.0950039], 56)
```

[14]:
```
k, norm(x-b1)
```

```
UndefVarError: b1 not defined



Stacktrace:

 [1] top-level scope at In[14]:1
```

We can compute the eigenvalues and condition number of $A$.

[15]:
```
eigen(A)
```

[15]:
```
Eigen{Float64,Float64,Array{Float64,2},Array{Float64,1}}
eigenvalues:
1000-element Array{Float64,1}:
 0.20001378984134793
 0.20005515922956107
 0.20012410775715758
 0.20022063474500001
 0.20034473924231072
 0.2004964200266737
 0.2006756756040492
 0.20088250420879217
```

```
 0.20111690380366315
 0.20137887207985267
 0.20166840645700262
 0.20198550408323332
 0.20233016183516794

 5.7980144959167665
 5.798331593542997
 5.798621127920147
 5.798883096196337
 5.799117495791208
 5.7993243243959505
 5.799503579973327
 5.79965526075769
 5.799779365255
 5.799875892242843
 5.799944840770439
 5.799986210158653
eigenvectors:
1000Œ1000 Array{Float64,2}:
 -0.000140286  -0.00028057   -0.000420851     0.00028057   0.000140286
  0.00028057    0.000561129   0.000841665      0.000561129  0.00028057
 -0.000420851  -0.000841665  -0.0012624        0.000841665  0.000420851
  0.000561129   0.00112217    0.00168303       0.00112217   0.000561129
 -0.0007014    -0.00140263   -0.00210351       0.00140263   0.0007014
  0.000841665   0.00168303    0.0025238       0.00168303   0.000841665
 -0.000981922  -0.00196337   -0.00294387       0.00196337   0.000981922
  0.00112217    0.00224363    0.00336368       0.00224363   0.00112217
 -0.0012624    -0.0025238    -0.00378319       0.0025238    0.0012624
  0.00140263    0.00280387    0.00420236       0.00280387   0.00140263
 -0.00154284   -0.00308384   -0.00462116      0.00308384   0.00154284
  0.00168303    0.00336368    0.00503955       0.00336368   0.00168303
 -0.00182321   -0.00364338   -0.0054575        0.00364338   0.00182321

 -0.00168303    0.00336368   -0.00503955      -0.00336368   0.00168303
  0.00154284   -0.00308384    0.00462116      -0.00308384   0.00154284
 -0.00140263    0.00280387   -0.00420236     -0.00280387   0.00140263
  0.0012624    -0.0025238     0.00378319      -0.0025238    0.0012624
 -0.00112217    0.00224363   -0.00336368      -0.00224363   0.00112217
  0.000981922  -0.00196337    0.00294387      -0.00196337   0.000981922
 -0.000841665   0.00168303   -0.0025238       -0.00168303   0.000841665
  0.0007014    -0.00140263    0.00210351     -0.00140263   0.0007014
 -0.000561129   0.00112217   -0.00168303      -0.00112217   0.000561129
  0.000420851  -0.000841665   0.0012624       -0.000841665  0.000420851
 -0.00028057    0.000561129  -0.000841665     -0.000561129  0.00028057
  0.000140286  -0.00028057    0.000420851     -0.00028057   0.000140286
```

[16]:
```
cond(A)
```

[16]:
```
28.997931666407833
```

Try to compute a simple precontionner using the inverse of the diagonal of matrix $A$.

[17]:
```
D = 1 ./diag(A)
M = Diagonal(D)
```

[17]:
```
1000Œ1000 Diagonal{Float64,Array{Float64,1}}:
 0.333333
          0.333333
                   0.333333
                            0.333333
                                        0.333333
                                                 0.333333
                                                          0.333333
```

Unfortunately, in this case, it does not help as the condition number is not improving.

[18]:
```
B = M*A
cond(B)
```

[18]:
```
28.997931666407865
```

Consider another situation when $A$ is diagonal.

[19]:
```
n = 1000;
A = zeros(n,n);
for i = 1:n
    A[i,i] = 10*rand()
end
b = zeros(n)
```

```
for i = 1:n
  b[i] = rand()
end
x0 = zeros(n)
cond(A)
```

[19]:
```
1371.965623700578
```

The solution we are looking for is

[20]:
```
A\b
```

[20]:
```
1000-element Array{Float64,1}:
 0.8862615969732116
 0.2296543135714681
 0.08190974579215711
 0.08575735751107477
 0.2182999826637982
 0.2374149948934806
 0.020092192319598193
 0.19806514122618815
 0.6957673526806002
 0.10001698295052926
 0.4816327167066336
 0.08790037345388463
 0.05066247295244987

 0.10077973559633926
 0.020086882003160982
 0.24277710470209954
 0.09021225081815908
 0.17700103234290007
 0.03335644650609706
 0.08842243991239374
 0.18012342025491074
 0.03708012565899532
 0.08022614848159804
 0.23341201083173646
 0.12877357422014438
```

Without preconditionning, with have the iterates sequence

[21]:
```
M = zeros(n,n)+I
x, iter, k = pcg_quadratic_tol(A, b, x0, M, true)
```

[21]:
```
([-0.886262, -0.229654, -0.0819097, -0.0857574, -0.2183, -0.237415, -0.0200922,
-0.198065, -0.695767, -0.100017    -0.242777, -0.0902123, -0.177001,
-0.0333564, -0.0884224, -0.180123, -0.0370801, -0.0802261, -0.233412,
-0.128774], Any[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0    0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], -0.0942439, -0.147646, -0.133322,
-0.100915, -0.108562, -0.189671, -0.0249358, -0.126222, -0.192307    -0.242777,
```

```
-0.0902123, -0.177001, -0.0333564, -0.0884224, -0.180123, -0.0370801,
-0.0802261, -0.233412, -0.128774], 219)
```

This is equivalent to the unpreconditioned version.

[22]:
```
x, iter, k = cg_quadratic_tol(A, b, x0, true)
```

```
UndefVarError: cg_quadratic_tol not defined

Stacktrace:

 [1] top-level scope at In[22]:1
```

However, since $A$ is diagonal, an obvious diagonal preconditionner is $A^{-1}$ itself.

[23]:
```
M = zeros(n,n)
for i = 1:n
    M[i,i] = 1/A[i,i]
end
```

The condition number of the preconditioned matrix is of course equal to 1.

[24]:
```
cond(M*A)
```

[24]:
```
1.0000000000000002
```

The theory then predicts that we converge in one iteration with the preciondititionned conjugate gradient.

[25]:
```
x, iter, k = pcg_quadratic_tol(A, b, x0, M, true)
```

[25]:
```
([-0.886262, -0.229654, -0.0819097, -0.0857574, -0.2183, -0.237415, -0.0200922,
-0.198065, -0.695767, -0.100017    -0.242777, -0.0902123, -0.177001,
-0.0333564, -0.0884224, -0.180123, -0.0370801, -0.0802261, -0.233412,
-0.128774], Any[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0    0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], -0.886262, -0.229654, -0.0819097,
-0.0857574, -0.2183, -0.237415, -0.0200922, -0.198065, -0.695767    -0.242777,
-0.0902123, -0.177001, -0.0333564, -0.0884224, -0.180123, -0.0370801,
-0.0802261, -0.233412, -0.128774], 1)
```

Consider now another example.

[26]:
```
A = zeros(n,n)+3*I
for i = 1:n-1
    A[i,i+1] = 1.4
    A[i+1,i] = 1.4
end
A
```

[26]:
```
1000Œ1000 Array{Float64,2}:
 3.0  1.4  0.0  0.0  0.0  0.0  0.0  0.0    0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
 1.4  3.0  1.4  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  1.4  3.0  1.4  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  1.4  3.0  1.4  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  1.4  3.0  1.4  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  1.4  3.0  1.4  0.0    0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  1.4  3.0  1.4     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  1.4  3.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.4     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0    0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0

 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0    0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     1.4  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     3.0  1.4  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     1.4  3.0  1.4  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0    0.0  1.4  3.0  1.4  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  1.4  3.0  1.4  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  1.4  3.0  1.4  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  1.4  3.0  1.4
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  1.4  3.0
```

[27]:
```
eigen(A)
```

[27]:
```
Eigen{Float64,Float64,Array{Float64,2},Array{Float64,1}}
eigenvalues:
1000-element Array{Float64,1}:
 0.20001378984134793
 0.20005515922956107
 0.20012410775715758
 0.20022063474500001
 0.20034473924231072
 0.2004964200266737
 0.2006756756040492
 0.20088250420879217
 0.20111690380366315
 0.20137887207985267
 0.20166840645700262
 0.20198550408323332
 0.20233016183516794

 5.7980144959167665
 5.798331593542997
 5.798621127920147
```

```
 5.798883096196337
 5.799117495791208
 5.7993243243959505
 5.799503579973327
 5.79965526075769
 5.799779365255
 5.799875892242843
 5.799944840770439
 5.799986210158653
eigenvectors:
1000×1000 Array{Float64,2}:
 -0.000140286  -0.00028057   -0.000420851      0.00028057    0.000140286
  0.00028057    0.000561129   0.000841665      0.000561129   0.00028057
 -0.000420851  -0.000841665  -0.0012624        0.000841665   0.000420851
  0.000561129   0.00112217    0.00168303       0.00112217    0.000561129
 -0.0007014    -0.00140263   -0.00210351       0.00140263    0.0007014
  0.000841665   0.00168303    0.0025238        0.00168303    0.000841665
 -0.000981922  -0.00196337   -0.00294387       0.00196337    0.000981922
  0.00112217    0.00224363    0.00336368       0.00224363    0.00112217
 -0.0012624    -0.0025238    -0.00378319       0.0025238     0.0012624
  0.00140263    0.00280387    0.00420236       0.00280387    0.00140263
 -0.00154284   -0.00308384   -0.00462116       0.00308384    0.00154284
  0.00168303    0.00336368    0.00503955       0.00336368    0.00168303
 -0.00182321   -0.00364338   -0.0054575        0.00364338    0.00182321

 -0.00168303    0.00336368   -0.00503955      -0.00336368    0.00168303
  0.00154284   -0.00308384    0.00462116      -0.00308384    0.00154284
 -0.00140263    0.00280387   -0.00420236      -0.00280387    0.00140263
  0.0012624    -0.0025238     0.00378319      -0.0025238     0.0012624
 -0.00112217    0.00224363   -0.00336368      -0.00224363    0.00112217
  0.000981922  -0.00196337    0.00294387      -0.00196337    0.000981922
 -0.000841665   0.00168303   -0.0025238       -0.00168303    0.000841665
  0.0007014    -0.00140263    0.00210351      -0.00140263    0.0007014
 -0.000561129   0.00112217   -0.00168303      -0.00112217    0.000561129
  0.000420851  -0.000841665   0.0012624       -0.000841665   0.000420851
 -0.00028057    0.000561129  -0.000841665     -0.000561129   0.00028057
  0.000140286  -0.00028057    0.000420851     -0.00028057    0.000140286
```

[28]:
```
A\(-b)
```

[28]:
```
1000-element Array{Float64,1}:
 -0.05568595774090107
 -0.21797963886902239
 -0.005652879196824504
 -0.24707652017354526
  0.17392067865618507
 -0.5141623521212706
  0.24900884950611296
```

```
-0.10867542080927135
-0.4678919917214
 0.42301839366527355
-0.7285540150474694
 0.5232803894722856
-0.5268028507225743

-0.4061064689877636
 0.24073782048113773
-0.16528525470435787
-0.10473494326723717
-0.2474516409293851
 0.24273344538640673
-0.39803080780471284
 0.0018418003203977324
-0.060982449487973295
-0.018683483375453114
-0.0674773610599925
-0.17003072785162449
```

[29]:
```julia
x, iter, k = cg_quadratic_tol(A, b, x0, true)
```

```
UndefVarError: cg_quadratic_tol not defined


Stacktrace:

 [1] top-level scope at In[29]:1
```

[30]:
```julia
M = zeros(n,n)
for i = 1:n
    M[i,i] = 1/A[i,i]
end
```

[31]:
```julia
cond(A)
```

[31]: 28.997931666407833

[32]:
```julia
cond(M*A)
```

[32]: 28.997931666407865

[33]:
```julia
x, iter, k = pcg_quadratic_tol(A, b, x0, M, true)
```

[33]:
```
([-0.055686, -0.21798, -0.00565288, -0.247077, 0.173921, -0.514162, 0.249009,
-0.108675, -0.467892, 0.423018    -0.165285, -0.104735, -0.247452, 0.242733,
-0.398031, 0.0018418, -0.0609824, -0.0186835, -0.0674774, -0.170031], Any[[0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0    0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

```
0.0, 0.0, 0.0, 0.0], -0.0923897, -0.144741, -0.130699, -0.0989292, -0.106426,
-0.185939, -0.0244452, -0.123739, -0.188523    -0.165285, -0.104735, -0.247452,
0.242733, -0.398031, 0.0018418, -0.0609824, -0.0186835, -0.0674774, -0.170031],
56)
```

There is no advantage.

[34]:
```
M = A^(-1)
```

[34]:
```
1000×1000 Array{Float64,2}:
  0.490553     -0.336899      0.231373        5.26731e-164  -2.45808e-164
 -0.336899      0.721926     -0.4958         -1.12871e-163   5.26731e-164
  0.231373     -0.4958        0.831055        1.89193e-163  -8.82901e-164
 -0.158901      0.340503     -0.570747       -2.92543e-163   1.3652e-163
  0.109129     -0.233848      0.391974        4.37685e-163  -2.04253e-163
 -0.0749471     0.160601     -0.269198       -6.45353e-163   3.01165e-163
  0.0514717    -0.110297      0.184878        9.45214e-163  -4.411e-163
 -0.0353494     0.0757488    -0.126969       -1.38011e-162   6.44049e-163
  0.0242771    -0.0520223     0.0871993       2.01216e-162  -9.39006e-163
 -0.0166729     0.0357276    -0.0598862      -2.93166e-162   1.36811e-162
  0.0114505    -0.0245368     0.0411283       4.26996e-162  -1.99265e-162
 -0.00786389    0.0168512    -0.0282458      -6.21827e-162   2.90186e-162
  0.00540072   -0.011573      0.0193985       9.05489e-162  -4.22562e-162

  2.90186e-162  -6.21827e-162   1.0423e-161     0.0168512    -0.00786389
 -1.99265e-162   4.26996e-162  -7.15727e-162   -0.0245368     0.0114505
  1.36811e-162  -2.93166e-162   4.91401e-162    0.0357276    -0.0166729
 -9.39006e-163   2.01216e-162  -3.37276e-162   -0.0520223     0.0242771
  6.44049e-163  -1.38011e-162   2.31332e-162    0.0757488    -0.0353494
 -4.411e-163     9.45214e-163  -1.58436e-162   -0.110297      0.0514717
  3.01165e-163  -6.45353e-163   1.08173e-162    0.160601     -0.0749471
 -2.04253e-163   4.37685e-163  -7.33643e-163   -0.233848      0.109129
  1.3652e-163   -2.92543e-163   4.90358e-163    0.340503     -0.158901
 -8.82901e-164   1.89193e-163  -3.17124e-163   -0.4958        0.231373
  5.26731e-164  -1.12871e-163   1.89193e-163    0.721926     -0.336899
 -2.45808e-164   5.26731e-164  -8.82901e-164   -0.336899      0.490553
```

[35]:
```
x, iter, k = pcg_quadratic_tol(A, b, x0, M, true)
```

[35]:
```
([-0.055686, -0.21798, -0.00565288, -0.247077, 0.173921, -0.514162, 0.249009,
-0.108675, -0.467892, 0.423018    -0.165285, -0.104735, -0.247452, 0.242733,
-0.398031, 0.0018418, -0.0609824, -0.0186835, -0.0674774, -0.170031], Any[[0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0    0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0], -0.055686, -0.21798, -0.00565288, -0.247077, 0.173921,
-0.514162, 0.249009, -0.108675, -0.467892    -0.165285, -0.104735, -0.247452,
0.242733, -0.398031, 0.0018418, -0.0609824, -0.0186835, -0.0674774, -0.170031],
1)
```

Consider now the following example.

[36]:
```
n = 1000
A = zeros(n,n)+Diagonal([2+i*i for i=1:n])
```

[36]:
```
1000Œ1000 Array{Float64,2}:
 3.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
 0.0  6.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0  11.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0  18.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0  27.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0  38.0        0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0

 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0    994011.0       0.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0  996006.0       0.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0  998003.0  0.0
 0.0  0.0   0.0   0.0   0.0   0.0         0.0       0.0       0.0  1.0e6
```

[37]:
```
for i = 1:n-1
    A[i,i+1] = 1
    A[i+1,i] = 1
end
A[n,1] = 1
A[1,n] = 1
cond(A)
```

[37]:
```
372201.88311699365
```

[38]:
```
 = cond(A)
(sqrt()-1)/(sqrt()+1)
```

[38]:
```
0.9967271248797949
```

[39]:
```
A
```

[39]:
```
1000Œ1000 Array{Float64,2}:
 3.0  1.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  1.0
```

```
1.0  6.0   1.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  1.0  11.0   1.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   1.0  18.0   1.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   1.0  27.0   1.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   1.0  38.0       0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   1.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0       0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0

0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0       0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0       1.0       0.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0   994011.0       1.0       0.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        1.0  996006.0       1.0  0.0
0.0  0.0   0.0   0.0   0.0   0.0        0.0       1.0  998003.0  1.0
1.0  0.0   0.0   0.0   0.0   0.0        0.0       0.0       1.0  1.0e6
```

[40]:
```
A^(-1)
```

[40]:
```
1000Œ1000 Array{Float64,2}:
  0.353263      -0.0597876     0.00546288      3.53969e-13  -3.53262e-7
 -0.0597876      0.179363     -0.0163886      -5.99071e-14   5.97875e-8
  0.00546288    -0.0163886     0.092869        5.4738e-15   -5.46287e-9
 -0.00030412     0.000912359  -0.00517004     -3.04728e-16   3.04119e-10
  1.12747e-5    -3.38241e-5    0.00019167      1.12972e-17  -1.12747e-11
 -2.96856e-7     8.90567e-7   -5.04654e-6     -2.97449e-19   2.96855e-13
  5.82243e-9    -1.74673e-8    9.89813e-8      5.83407e-21  -5.82242e-15
 -8.82347e-11    2.64704e-10  -1.49999e-9     -8.84111e-23   8.82346e-17
  1.06319e-12   -3.18958e-12   1.80743e-11     1.06532e-24  -1.06319e-18
 -1.04243e-14    3.12729e-14  -1.77213e-13    -1.04451e-26   1.04243e-20
  8.47552e-17   -2.54265e-16   1.44084e-15     8.49246e-29  -8.4755e-23
 -5.80538e-19    1.74161e-18  -9.86915e-18    -5.81699e-31   5.80537e-25
  3.39506e-21   -1.01852e-20   5.7716e-20      3.40185e-33  -3.39505e-27

  4.03306e-73   -6.82572e-74   6.23676e-75     1.14166e-66  -1.14166e-72
 -3.94483e-67    6.67639e-68  -6.10032e-69    -1.11669e-60   1.11669e-66
  3.86634e-61   -6.54355e-62   5.97894e-63     1.09447e-54  -1.09447e-60
 -3.79706e-55    6.4263e-56   -5.87181e-57    -1.07486e-48   1.07486e-54
  3.73656e-49   -6.32391e-50   5.77825e-51     1.05773e-42  -1.05773e-48
```

```
 -3.68444e-43   6.2357e-44   -5.69765e-45     -1.04298e-36   1.04298e-42
  3.64037e-37  -6.1611e-38    5.62949e-39      1.0305e-30   -1.0305e-36
 -3.60406e-31   6.09966e-32  -5.57335e-33    -1.02022e-24   1.02022e-30
  3.57529e-25  -6.05097e-26   5.52887e-27      1.01208e-18  -1.01208e-24
 -3.55388e-19   6.01473e-20  -5.49575e-21     -1.00602e-12   1.00602e-18
  3.53969e-13  -5.99071e-14   5.4738e-15       1.002e-6     -1.002e-12
 -3.53262e-7    5.97875e-8   -5.46287e-9      -1.002e-12     9.99998e-7
```

[41]:

```
M = zeros(n,n)+I
x, iter, k = pcg_quadratic_tol(A, b, x0, M, true)
```

[41]:

```
([-0.118137, -0.100799, -0.0459062, -0.0256695, -0.0191955, -0.0239089,
-0.00206028, -0.00988785, -0.0111027, -0.00411876    -1.8521e-7, -9.0649e-7,
-5.56925e-7, -1.77638e-7, -9.39576e-7, -6.42465e-7, -2.0785e-7, -3.99732e-7,
-4.67694e-7, -9.07648e-7], Any[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], -1.4667e-6, -2.2978e-6,
-2.07487e-6, -1.57052e-6, -1.68953e-6, -2.95181e-6, -3.88072e-7, -1.96437e-6,
-2.99284e-6    -1.8521e-7, -9.0649e-7, -5.56925e-7, -1.77638e-7, -9.39576e-7,
-6.42465e-7, -2.0785e-7, -3.99732e-7, -4.67694e-7, -9.07648e-7], 1001)
```

[42]:

```
M = zeros(n,n)
for i = 1:n
    M[i,i] = 1/A[i,i]
end
cond(A*M), cond(A)
```

[42]:

```
(1.926360732450869, 372201.88311699365)
```

[43]:

```
M
```

[43]:

```
1000Œ1000 Array{Float64,2}:
 0.333333  0.0       0.0            0.0         0.0       0.0
 0.0       0.166667  0.0             0.0         0.0       0.0
 0.0       0.0       0.0909091       0.0         0.0       0.0
 0.0       0.0       0.0             0.0         0.0       0.0
 0.0       0.0       0.0             0.0         0.0       0.0
 0.0       0.0       0.0            0.0         0.0       0.0
 0.0       0.0       0.0             0.0         0.0       0.0
 0.0       0.0       0.0             0.0         0.0       0.0
 0.0       0.0       0.0             0.0         0.0       0.0
 0.0       0.0       0.0             0.0         0.0       0.0
 0.0       0.0       0.0            0.0         0.0       0.0
 0.0       0.0       0.0             0.0         0.0       0.0
 0.0       0.0       0.0             0.0         0.0       0.0

 0.0       0.0       0.0             0.0         0.0       0.0
 0.0       0.0       0.0             0.0         0.0       0.0
 0.0       0.0       0.0            0.0         0.0       0.0
 0.0       0.0       0.0             0.0         0.0       0.0
 0.0       0.0       0.0             0.0         0.0       0.0
```

```
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0          0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0           1.00401e-6  0.0        0.0
 0.0       0.0       0.0           0.0         1.002e-6   0.0
 0.0       0.0       0.0           0.0         0.0        9.99998e-7
```

[48]:

```
A*M
```

[48]:

```
1000×1000 Array{Float64,2}:
 1.0       0.166667  0.0          0.0         0.0        9.99998e-7
 0.333333  1.0       0.0909091     0.0         0.0        0.0
 0.0       0.166667  1.0           0.0         0.0        0.0
 0.0       0.0       0.0909091     0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0          0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0          0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0

 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0          0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0           0.0         0.0        0.0
 0.0       0.0       0.0          0.0         0.0        0.0
 0.0       0.0       0.0           1.00401e-6  0.0        0.0
 0.0       0.0       0.0           1.0         1.002e-6   0.0
 0.0       0.0       0.0           1.00401e-6  1.0        9.99998e-7
 0.333333  0.0       0.0           0.0         1.002e-6   1.0
```

[51]:

```
x, iter, k = pcg_quadratic_tol(A, b, x0, M, true)
```

[51]:

```
([-0.126091, -0.0939571, -0.0499804, -0.0242963, -0.0183406, -0.0244805,
-0.00178595, -0.00938221, -0.0114508, -0.00379961    -3.11023e-7, -9.06479e-7,
-5.56924e-7, -1.77598e-7, -8.60266e-7, -6.42218e-7, -2.07769e-7, -2.36839e-7,
-4.67565e-7, -4.78468e-7], Any[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], -0.131425, -0.102948,
-0.0507055, -0.0234546, -0.0168213, -0.0208816, -0.0020455, -0.00800088,
-0.00969313    -3.11023e-7, -9.06479e-7, -5.56924e-7, -1.77598e-7, -8.60266e-7,
-6.42218e-7, -2.07769e-7, -2.36839e-7, -4.67565e-7, -4.78468e-7], 7)
```

```
[ ]:
```