6.1 Steepest Descent Method

As motivation consider the membrane problem with small deformation and in a steady state so that the potential energy in a minimum.

Potential Energy $\cong T\left(\sqrt{1+u_x^2+u_y^2}-1\right)dxdy - fu\Delta x\Delta y$

= surface tension + external work , where

T = tension,

u = deformation and

f is external pressure on the membrane.

Use $f(p)\equiv\sqrt{1+p}\cong f(0) + f'(0)p$

$= 1 + \frac{1}{2}\frac{1}{\sqrt{1+p}}\Big|_{p=0}(p-0) = 1 + 1/2\ p.$

Let $p = \left(u_x^2+u_y^2\right)$ so that

$$\sqrt{1+u_x^2+u_y^2}-1\cong\frac{1}{2}\left(u_x^2+u_y^2\right)$$

$P(u) = \iint_\Omega\left(\frac{1}{2}\left(u_x^2+u_y^2\right)- fu\right)dxdy$ is an approximation of the total potential energy.

One can show the following are equivalent formulations:

1. Potential Energy

   P(u) = min P(v) where v is in a "suitable" set of functions, S.

2. Weak Form

$$\iint T\left(u_x j_x + u_y j_y\right) - \iint f j = 0, \text{ for all "suitable" } j$$

3. Classical Form

$$-\mathrm{T}\left(u_{xx} + u_{yy}\right) = f .$$

For example, to show a potential energy solution is a weak solution, use $u + l j$ in P(u) so that $f(\lambda) = \mathrm{P}\left(u + l j\right)$ is a function of the real number λ. Because u minimizes the potential energy, λ = 0 will minimize f(λ) so that f '(0) = 0, which corresponds to the weak equation.

Consider Ax = d where A is SPD and is from the classical form. The linear system is related to the potential energy form where

$\mathrm{J}(x) = \frac{1}{2} x^T A x - x^T d$, from J(x) comes from the potential energy

Algebraic Lemma. $\mathrm{J}(y) = \mathrm{J}(x) + \frac{1}{2}(y-x)^T A(y-x) - (y-x)^T r(x)$

Proposition 1. If A is SPD, then 1 and 2 are equivalent

1. Ax = d,
2. J(x) = min J(y).

Proof of Lemma.

Let $y = x + (y - x)$.
Use $A = A^T$.

$$J(y) = J(x + y - x)$$

$$= \frac{1}{2}\left(x + (y-x)\right)^T A(x + (y-x)) - (x + (y - x))^T d$$

$$= \frac{1}{2}x^T Ax + \frac{1}{2}(y-x)^T Ax + \frac{1}{2}x^T A(y-x) + \frac{1}{2}(y-x)^T A(y-x) - x^T d - (y-x)^T d$$

$$= J(x) + (y-x)^T Ax - (y-x)^T d + \frac{1}{2}(y-x)^T A(y-x)$$

$$= J(x) - (y-x)^T r(x) + \frac{1}{2}(y-x)^T A(y-x).$$

Proof 1 implies 2:

Ax = d means r(x) =0.

Use the Algebraic Lemma, A being SPD and r(x) = 0 to get

$J(y) = J(x) + \frac{1}{2}(y\text{-}x)^T A(y\text{-}x) - 0 \geq J(x).$

Proof 2 implies 1:

We want to show r(x) = 0, that is, $[r(x)]_i = 0$.

This is equivalent to showing $[r(x)]_i \geq 0$ and $[r(x)]_i \leq 0$.

Since y is arbitrary,

y = x + (y - x) and choose y so that

$y - x \equiv \lambda e_i$.

$J(y) = J(x + \lambda e_i)$, by the Algebraic Lemma

$= J(x) + \frac{1}{2}(\lambda e_i)^T A(\lambda e_i) - (\lambda e_i)^T r(x)$

$$= J(x) + \frac{1}{2}\lambda^2 a_{ii} - \lambda[r(x)]_I$$

$$0 \le J(y) - J(x) = \lambda(\frac{1}{2}\lambda a_{ii} - [r(x)]_i)$$

Since A is SPD, $a_{ii} > 0$.

(a) Suppose $[r(x)]_i < 0$,

Let $\lambda \uparrow 0$.

So eventually $(\frac{1}{2}\lambda a_{ii} - [r(x)]_i) > 0$,

This implies $J(y) - J(x) < 0$, which is a contradiction.

Therefore, we must have $[r(x)]_i \ge 0$.

(b) Suppose $[r(x)]_i > 0$,

Let $\lambda \downarrow 0$. So eventually $(\frac{1}{2}\lambda a_{ii} - [r(x)]_i) < 0$, and this implies

$J(y) - J(x) < 0$, which is a contradiction.

Then we must have $[r(x)]_i \le 0$.

Idea for Steepest Descent Method:

Let $f(\alpha) = J(x^0 + \alpha p)$, and p be some direction. We want to choose α so that $f(\alpha)$ the smallest possible. This is a simpler problem because f is a function of a single variable. In order to choose the direction p so that the directional derivative of J(x) is the largest possible, we will need the following results.

Proposition 2. 1. Cauchy Inequality.

$$|x^T y| \le \|x\|_2 \|y\|_2$$

2. Directional Derivative.

$$\frac{df}{du} = \nabla f \cdot u \quad , \text{where } u^T u = 1, \ \frac{df}{du} \equiv \lim_{t \to 0} \frac{f(x+tu) - f(x)}{t}$$

3. Direction of Steepest Descent.

$$\max_u \left| \frac{df}{du} \right| = \|\nabla f\|_2 \quad \text{when } u \equiv \nabla f / \|\nabla f\|_2$$

Proof of 1 :

$$0 \le f(t) \equiv (x + ty)^T (x + ty)$$

$$= x^T x + t2\, x^T y + t^2\, y^T y \qquad , \text{because } x^T y = y^T x.$$

$f'(t_1) = 0$ implies $t_1 = - x^T y / y^T y$.

$f''(t_1) = 2\, y^T y > 0$, so $f(t_1)$ is the min. of $f(t)$.

$$0 \le f(t_1) = x^T x + 2(-x^T y / y^T y)(x^T y) + (- x^T y / y^T y)^2 (y^T y)$$

$$= x^T x - (x^T y)^2 / y^T y$$

This implies $(x^T y)^2 \le (x^T x)(y^T y) = (\|x\|_2 \ \|y\|_2)^2$.

Proof of 2 :

$$\lim_{t \to 0} \frac{f(x+tu) - f(x)}{t} = \lim_{t \to 0} \left(\frac{f(x + t(u_1, u_2, .., u_n)) - f(x + t(0, u_2, ..., u_n))}{t u_1} \right. u_1$$

$$+ \frac{f(x+t(0,u_2,..,u_n)) - f(x+t(0,0,u_3,...,u_n))}{tu_2} u_2$$

$$+ .....$$

$$+ \frac{f(x+t(0,...,0,u_n)) - f(x+t(0,0,...,0))}{tu_n} u_n)$$

$$= f_{x1}u_1 + ... + f_{xn}u_n = \nabla f \cdot u$$

Proof of 3:

$$|\frac{df}{du}| = |\nabla f \cdot u| \leq \|\nabla f\|_2 \cdot \|u\|_2 = \|\nabla f\|_2 \ 1$$

Because $\|u\|_2^2 = (\nabla f / \|\nabla f\|_2)^T (\nabla f / \|\nabla f\|_2) = 1$,

we may choose $u = \nabla f / \|\nabla f\|_2$. Then for this u

$$\frac{df}{du} = \nabla f \cdot u = \nabla f \cdot (\nabla f / \|\nabla f\|_2) = \|\nabla f\|_2.$$

Therefore, the largest possible $|\frac{df}{du}|$ is given by this u.

6.2 Steepest Descent Algorithm in Multiple Directions

Consider $J(x^0 + \alpha p)$. We want to choose α and p so that this is the smallest possible. This is a simpler problem because α is a single number, and p is a direction so that J should decrease most rapidly.

Proposition 3. If A is symmetric, then the direction of steepest descent is

$\nabla J = -r(x)$.

Proof.

$$
\begin{aligned}
[\nabla J]_i &= \frac{\partial}{\partial x_{\hat{i}}}\left(\frac{1}{2}x^T Ax - x^T d\right) \\
&= \frac{\partial}{\partial x_{\hat{i}}}\left(\frac{1}{2}\sum_{i,j} x_i a_{ij} x_j - \sum_i x_i d_i\right) \\
&= \frac{1}{2}\sum_{i,j}\frac{\partial}{\partial x_{\hat{i}}} x_i a_{ij} x_j - \sum_i \frac{\partial}{\partial x_{\hat{i}}} x_i d_i \\
&= \frac{1}{2}\sum_j 1\, a_{\hat{i}j} x_j + \frac{1}{2}\sum_i x_i a_{i\hat{i}} 1 - 1\, d_{\hat{i}}, \quad \text{use } a_{\hat{i}i} = a_{i\hat{i}} \\
&= \sum_j a_{\hat{i}j} x_j - d_{\hat{i}} \\
&= [-r(x)]_{\hat{i}}.
\end{aligned}
$$

Proposition 4. If A is SPD, then

$$J(x^+) = \min_a J(x + ar) \text{ where}$$

$$x^+ = x + \hat{a}r \text{ and } \hat{a} = \frac{r^T r}{r^T A r}.$$

Proof. Let $f(\alpha) = J(x + \alpha p)$ where $p = -r$, and use the Algebraic Lemma to get

$f(\alpha) = J(x) + 1/2\ \alpha^2\ p^T Ap - \alpha p^T r$.

Then $f'(\alpha) = 0 = \alpha\, p^T Ap - p^T r$, and note $p^T Ap > 0$ for nonzero p.

Thus for $p = -r$ we have $\alpha = -r^T r / r^T Ar$ and $x^+ = x + \alpha(-r)$.

Or, $x^+ = x + (r^T r / r^T Ar)\, r$.

Steepest Descent Algorithm.

x^o = initial guess

for m = 0, maxm

$r_m = d - Ax^m$

$\alpha = r_m{}^T r_m \;/\; r_m{}^T Ar_m$

$x^{m+1} = x^m + \alpha\, r_m$

test for convergence.

The next residual r_{m+1} may be computed using the previous residual:

$$r_{m+1} = d - A\,x^{m+1} = d - A(x^m + \alpha r_m) = d - Ax^m - \alpha Ar_m = r_m - \alpha Ar_m .$$

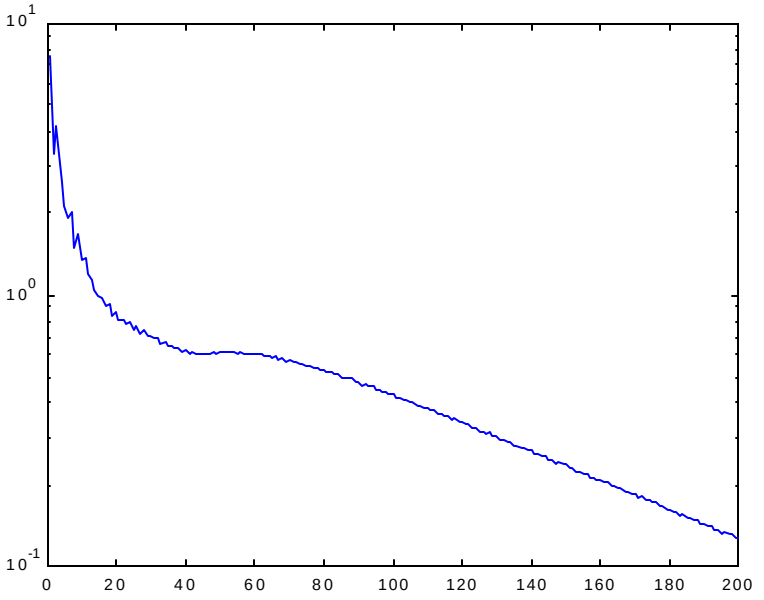
Thus, each iteration of the steepest descent algorithm requires one matrix-vector product, two dotproducts and one vector update.

Consider the partial differential equation $-u_{xx} - u_{yy} = f(x,y)$ where u must be equal to zero on the boundary of the unit square. In the Matlab code observe the use of array operations. The vectors are represented as 2D arrays, and the sparse matrix A is not explicitly stored. The product Ar is stored in the 2D array q. Here the partial differential equation has right side equal to $200 + 200\sin(\pi x)\sin(\pi y)$, and the solution is required to be zero on the boundary of $(0,1)x(0,1)$. The steepest descent method appears to be

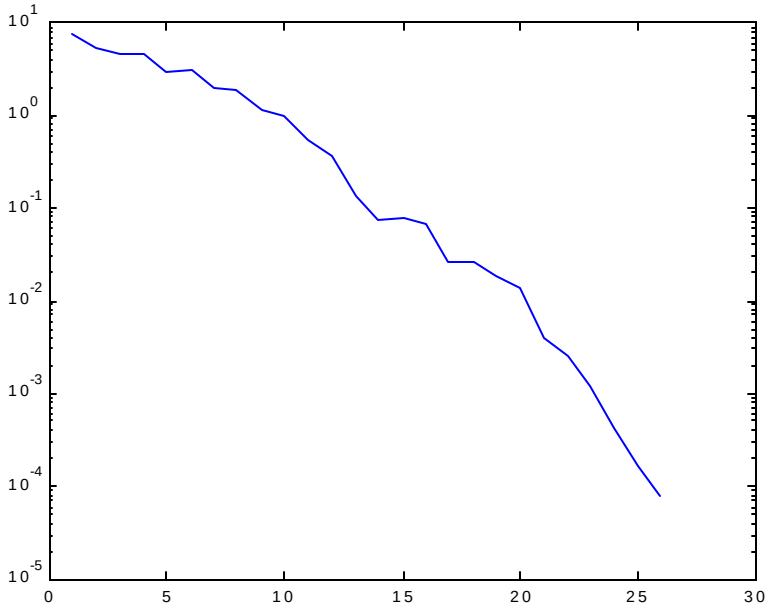
converging, but after 200 iterations the norm of the residual is still only about 10^{-1}. In the next section the conjugate gradient method will be described. One calculation is included here and shows that after only 26 iterations of the conjugate gradient method, the norm of the residual is about 10^{-4}. It is interesting to note if the right side is $200\sin(\pi x)\sin(\pi y)$, then the steepest descent method will converge in one iteration….Why?

Matlab Steepest Descent Code (st.m)

```
clear;
n = 20;
h = 1./n;
u(1:n+1,1:n+1)= 0.0;
r(1:n+1,1:n+1)= 0.0;
r(2:n,2:n)= 1000.*h*h;
for j= 2:n
   for i = 2:n
      r(i,j)= h*h*200*(1+sin(pi*(i-1)*h)*sin(pi*(j-1)*h));
   end
end
q(1:n+1,1:n+1)= 0.0;
err = 1.0;
m = 0;
rho = 0.0;
while ((err>.0001)*(m<200))
   m = m+1;
   oldrho = rho;
   rho = sum(sum(r(2:n,2:n).^2));
   for j= 2:n
      for i = 2:n
         q(i,j)=4.*r(i,j)-r(i-1,j)-r(i,j-1)-r(i+1,j)-r(i,j+1);
      end
   end
   alpha = rho/sum(sum(r.*q));
   u = u + alpha*r;
   r = r - alpha*q;
   err = max(max(abs(r(2:n,2:n))));
   reserr(m) = err;
end
m
semilogy(reserr)
```



Log(norm(r)) versus m for the Steepest Descent Method



Log(norm(r)) versus m for the Conjugate Gradient Method

Steepest Descent with Multiple Directions.

The steepest descent method computes the smallest J(x) for each direction:

$$x^1 = x^0 + a_0 r_0$$

$$x^2 = x^1 + a_1 r_1 = x^0 + a_0 r_0 + a_1 r_1 .$$

In order to obtain smaller values of J(x), we may minimize over larger dimensional sets of functions given by multiple directions:

$$x^1 = x^0 + c_0 r_0$$

$$x^2 = x^1 + c_0 r_0 + c_1 r_1 \text{ (use } \textbf{two} \text{ directions).}$$

Next, c_0 and c_1 now will be found so that

$$\min_{c_0, c_1} f(c_0, c_1) \quad \text{where} \quad f(c_0, c_1) \equiv J(x^1 + c_0 r_0 + c_1 r_1).$$

In general, we consider m+1 directions

$$x^{m+1} = x^m + c_0 r_0 + \cdots + c_m r_m$$

$$f(c_0, \cdots, c_m) \equiv J(x^m + c_0 r_0 + \cdots + c_m r_m)$$

Find $c = (c_0, \ldots, c_m)$ so that $\min_c f(c)$. Use the vector notation so that

$$x_{m+1} = x_m + \begin{bmatrix} r_0 & \cdots & r_m \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_m \end{bmatrix} = x_m + \overset{n \times (m+1)}{R} \quad \overset{(m+1) \times m}{c} .$$

Find c so that $f(c) = J(x^m + Rc)$ is a minimum.

Proposition 5. If A is SPD, and R has full column rank, then c such that f(c) is minimum is given by $(R^T AR)c = R^T r_m$.

Proof. Again use the Algebraic Lemma to obtain

$$J(x_{m+1}) = J(x_m) + \frac{1}{2}(Rc)^T A(Rc) - (Rc)^T r_m .$$

Define

$$\hat{J}(c) \equiv \frac{1}{2}(Rc)^T A(Rc) - (Rc)^T r_m$$
$$= \frac{1}{2} c^T \hat{A} c - R^T \hat{d},$$

where $\hat{A} \equiv R^T AR$ is SPD and $\hat{d} \equiv R^T r_m$.

$\hat{A} \equiv R^T AR$ is SPD because A is SPD and R has full column rank. Use the equivalence, given by Proposition 1, of the minimum of $\hat{J}(c)$ and the solution of $\hat{A}c = \hat{d}$.

One difficulty with this is that as m gets large more computations must be done to find $R^T AR = [r_i{}^T A\, r_j]$ and then to solve $(R^T AR)\, c = R^T r_m$. If the residuals were orthogonal with respect to the inner product given by A, then the matrix $R^T AR$ would be diagonal. The conjugate gradient method uses a version of the Gram-Schmidt process to ensure this is the case.

6.3 Conjugate Gradient Method

In order to simplify the solution of $(R^T AR)c = R^T r_m$, we will apply the Gram-Schmidt process to the residuals and use the inner product given by the SPD matrix, A. This will convert the matrix $(R^T AR)$ into a diagonal matrix.

Two directions m = 1:

$p_0 \equiv r_0$

$p_1 \equiv r_1 + \beta p_0$

Choose β so that $(p_1, p_0)_A = 0$

$(r_1 + \beta p_0)^T A p_0 = 0$

$r_1^T A p_0 + \beta p_0^T A p_0 = 0$

So, $\beta = \dfrac{-r_1^T A p_0}{p_0^T A p_0}$, where the $p_0^T A p_0 > 0$ because A is SPD.

$$x^2 = x^1 + c_0 p_0 + c_1 p_1$$
$$= x^1 + c_0 p_0 + c_1 (r_1 + \beta p_0)$$
$$= x^1 + \hat{c}_0 r_0 + \hat{c}_1 r_1$$

Choose c_0 and c_1 so that

$J(x^1 + c_0 p_0 + c_1 p_1)$ is a minimum.

This true if and only if

$P^T A P c = P^T r_1$ where

$P = [p_0 \; p_1]$ and

$c = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$.

Or,

$$\begin{bmatrix} p_0^T A p_0 & p_0^T A p_1 \\ p_1^T A p_0 & p_1^T A p_1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} p_0^T r_1 \\ p_1^T r_1 \end{bmatrix}.$$

Proposition 6. Let A be SPD.

If (a). $p_0 = r_o$,

(b). $p_1 = r_1 + \beta_0 p_0$ where $\beta_0 = -r_1^T Ap_0/p_0^T Ap_0$,

(c). $x^1 = x^0 + \alpha_0 r_0$ where $\alpha_0 = r_0^T r_0/r_0^T Ar_0$,

then

1. $p_1^T Ap_0 = 0$,
2. $p_0^T r_1 = 0$, and hence, $c_0 = 0$,
3. $p_1^T r_1 = r_1^T r_1$, and hence, $c_1 = r_1^T r_1/p_1^T Ap_1 = \alpha_1$ and
4. $\beta_0 = r_1^T r_1/ r_0^T r_0$.

Overall, $p_1 = r_1 + \beta_0 p_0$ and $x^2 = x^1 + \alpha_1 p_1$.

Proof of 1. By definition of β.

Proof of 2.

$$\begin{aligned} r_1 &= r(x_1) = d - Ax_1 \\ &= d - A(x_0 + a_0 p_0) \\ &= (d - Ax_0) - a_0 Ap_0 \\ &= r_0 - a_0 Ap_0 \end{aligned}$$

$$\begin{aligned} p_0^T r_1 &= p_0^T (r_0 - a_0 Ap_0) \\ &= r_0^T r_0 - a_0 p_0^T Ap_0 \\ &= 0. \end{aligned}$$

Proof of 3.

$$\begin{aligned} p_1^T r_1 &= (r_1 + b_0 p_0)^T r_1 \\ &= r_1^T r_1 + b_0 p_0^T r_1 \\ &= r_1^T r_1 + b_0\ 0. \end{aligned}$$

Proof of 4.

$$r_1^T r_1 = (r_0 - a\,Ar_0)^T (r_0 - a\,Ar_0)$$

$$= -r_0^T r_0 + r_0^T r_0 \frac{r_0^T r_0}{(r_0^T Ar_0)^2} (Ar_0)^T (Ar_0)$$

$$r_1^T Ar_0 = (r_0 - a\,Ar_0)^T Ar_0$$

$$= r_0^T Ar_0 - \frac{r_0^T r_0}{r_0^T Ar_0} (Ar_0)^T (Ar_0)$$

Thus, $b_0 = -\frac{r_1^T Ar_0}{r_0^T Ar_0} = \frac{r_1^T r_1}{r_0^T r_0}$.

Use three directions m=2:

$$x^3 = x^2 + c_0 p_0 + c_1 p_1 + c_3 p_3$$

$$p_0 \equiv r_0$$

$$p_1 \equiv r_1 + b_0 p_0$$

$$p_2 \equiv r_2 + b_1 p_1$$

Choose b_1 so that $(p_2, p_1)_A = 0$

$$(r_2 + b_1 p_1)^T Ap_1 = 0$$

$$b_1 = \frac{-r_2^T Ap_1}{p_1^T Ap_1}.$$

$\min J(x^2 + c_0 p_0 + c_1 p_1 + c_2 p_2)$ if and only if $P^T APc = P^T r_2$.

Or,

$$\begin{bmatrix} p_0^T Ap_0 & 0 & p_0^T Ap_2 \\ 0 & p_1^T Ap_1 & 0 \\ p_2^T Ap_0 & 0 & p_2^T Ap_2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} p_0^T r_2 \\ p_1^T r_2 \\ p_2^T r_2 \end{bmatrix}.$$

Fortunately, we can show

$p_0^T r_2 = p_1^T r_2 = 0$ and $p_0^T Ap_2 = p_2^T Ap_0 = 0$ so that

$x^3 = x^2 + 0p_0 + 0p_1 + c_2 p_2$ where

$$c_2 = \frac{p_2^T r_2}{p_2^T Ap_2} = \frac{r_2^T r_2}{p_2^T Ap_2} = a_2.$$

Proposition 7. Let A be SPD and m = 2.

If (a). Let p_0, x^1, p_1 be defined as in Proposition 6.

(b). $p_2 = r_2 + b_1 p_1$

$$b_1 = \frac{-r_2^T A p_1}{p_1^T A p_1}$$

(c). $x^2 = x^1 + a_1 p_1$

$$a_1 = \frac{r_1^T r_1}{p_1^T A p_1},$$

then

1. $p_0^T r_2 = 0$
2. $p_1^T r_2 = 0$
3. $p_1^T A p_2 = 0$
4. $p_0^T A p_2 = 0$
5. $p_2^T r_2 = r_2^T r_2$,and hence, $c_2 = \frac{r_2^T r_2}{p_2^T A p_2} = a_2$
6. $b_1 = \frac{r_2^T r_2}{r_1^T r_1}$.

Overall, $p_2 = r_2 + b_1 p_1$ and $x^3 = x^2 + a_2 p_2$.

Proof of 1.

$$r_2 = r_1 - c_1 A p_1$$

$$\begin{aligned} p_0^T r_2 &= r_0^T (r_1 - c_1 A p_1) \\ &= r_0^T r_1 - c_1 r_0^T A p_1 \quad , r_0^T A p_1 = 0 \\ &= r_0^T (r_0 - c_0 A r_0) \\ &= 0 . \end{aligned}$$

Conjugate Gradient Method.

$x^{m+1} = x^m + a\, p_m$ (a *represents the steepest descent formula.*)

$p_{m+1} = r_{m+1} + b\, p_m$ (b *repesents the "conjugate" direction* $(p_{m+1}, p_m)_A = 0.$)

$\min J(x^{m+1})$ if and only if $P^T APc = P^T r_m$

$$\begin{bmatrix} p_0{}^T A p_0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & p_m{}^T A p_m \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ p_m{}^T r_m \end{bmatrix}$$

Preconditioned Conjugate Gradient Algorithm (M = I is Conjugate Gradient).

choose x_0

solve $M\hat{r}_0 = r_0$ and $p_0 = \hat{r}_0$

for $m = 0, \max m$

$$a_m = \frac{\hat{r}_m{}^T r_m}{p_m{}^T A p_m} \qquad (\textit{steepest descent})$$

$x^{m+1} = x^m + a_m p_m$

$r_{m+1} = r_m - a_m A p_m$

test for convergence

solve $M\hat{r}_{m+1} = r_{m+1}$ (*preconditioning*)

$$b_m = \frac{\hat{r}_{m+1}{}^T r_{m+1}}{\hat{r}_m{}^T r_m}$$

$p_{m+1} = \hat{r}_{m+1} + b_m p_m.$ (*conjugatedirection*)

6.4 Preconditioned Conjugate Gradient

Error for the CG is a function of the condition number of A, $k_2(A) = \frac{\max|l|}{\min|l|}$.

The fastest convergence of the CG method occurs when $k_2(A) \approx 1$. Preconditioning can

Be viewed as finding an equivalent $\hat{A}\hat{X} = \hat{d}$ such that

$$K_2(\hat{A}) - 1 < K_2(A) - 1$$

There are three equivalent descriptions of the CG scheme:

1. $J(x^{m+1}) = \min_C J(x^m + c_o r_o + ... + c_m r_m)$

   where r_i are residual directions,

2. $J(x^{m+1}) = \min_C J(x^m + c_o p_o + ... + c_m p_m)$

   where p_i are conjugate directions, and

3. $J(x^{m+1}) = \min_C J(x^o + c_o r_o + c_1 A r_o ..... + c_m A^m r_o)$

   where $A^i r_0$ are Krylov directions.

Proposition 8. If A is SPD, then 1,2 and 3 are equivalent.

Proof.

$1 \leftrightarrow 2$, see formal proof on Stoer and Bulrich.

$2 \leftrightarrow 3$, see Kelley.

Connection among 1,2,3:

Let p_i be the conjugate directions as defined in the conjugate gradient algorithm.

$$p_o \equiv r_o$$

$$x^1 = x^o + a_o p_o$$

$$r_1 = r_o - a_o A p_o$$
$$p_1 = r_1 + b_o p_o$$
$$x^2 = x^1 + a_1 p_1$$
$$= x^1 + a_1 (r_1 + b_o p_o)$$
$$= x^1 + a_1 (r_1 + b_o r_o) \text{ , residual directions}$$
$$= x^o + a_o r_o + a_1 (r_o - a_o A p_o + b_o r_o)$$
$$= x^o + c_o r_o + c_1 A r_o \text{ , Krylov directions.}$$

Proposition 9. If A is SPD, then

$$\left\| x^{m+1} - x \right\|_A \le 2 \left(\frac{\sqrt{k_2} - 1}{\sqrt{k_2} + 1} \right)^{m+1} \left\| x^o - x \right\|_A$$

where Ax = d, $k_2 = k_2(A)$ and $\left\| x \right\|_A^2 = x^T A x$.

"Outline of proof"

Use the Algebraic Lemma

$$J(x^{m+1}) - J(x) = 1/2 (x^{m+1} - x)^T A (x^{m+1} - x)$$

$$= 1/2 \left\| x^{m+1} - x \right\|_A^2 .$$

$$x - x^{m+1} = x - (x^o + c_o r_o + \ldots\ldots\ldots c_m A^m r_o)$$
$$= x - x^o - (c_o r_o + \ldots\ldots\ldots c_m A^m r_o)$$
$$= x - x^o - (c_o I + c_1 A + \ldots\ldots\ldots c_m A^m) r_o$$

$$r_o = d - A x^o$$
$$= Ax - A x^o$$
$$= A(x - x^o)$$

$$x - x^{m+1} = x - x^o - (c_o I + c_1 A + \ldots\ldots\ldots c_m A^m) r_o A (x^o - x)$$

$$= (I - (c_o I + c_1 A + ...c_m A^m)A)(x^o - x)$$

So, by the Algebraic Lemma

$$2(J(x^{m+1}) - J(x)) = \left\| x^{m+1} - x \right\|_A^2 \le \left\| q_m(A)(x - x^o) \right\|_A^2 \quad \text{where}$$

$$q_m(z) = 1 - (c_o z + ...+ c_m z^{m+1}).$$

To obtain an error estimate choose a "good" polynomial $q_m(z)$.

Form of Preconditioner.

$A = M - N$
M is SPD
$M^{-1} = S^T S$
$Ax = d$
$M^{-1}Ax = M^{-1}d$
$S^T SAx = S^T Sd$
$S^T SAS^T (S^{-T}x) = S^T Sd$
$(SAS^T)(S^{-T}x) = Sd$
Let $\hat{A} = SAS^T$
$\hat{x} = S^{-T}x$
$\hat{d} = Sd,$

Apply CG to $\hat{A}\hat{x} = \hat{d}$ and use the definition $M^{-1} = S^T S$ to get the PCG .

Examples.

1. M = diagonal part of A

   or

   = block diagonal part of A
2. M = incomplete Cholesky factorization
3. M = incomplete domain decomposition

4. M for symmetric SOR splitting as follows:

$$\text{Let } w = 1.$$

$$A = D - L - L^T$$

$$(D - L)x^{m+1/2} = d + L^T x^m \qquad \text{Forward SOR}$$

$$(D - L^T)x^{m+1} = d + Lx^{m+1/2} \qquad \text{Backward SOR}$$

$$= d + L(D - L)^{-1}(d + L^T x^m)$$

$$x^{m+1} = (D - L^T)^{-1}[d + L(D - L)^{-1}(d + L^T x^m)]$$

$$= (D - L^T)^{-1} d + (D - L^T)^{-1} L(D - L)^{-1} d + (D - L^T)^{-1} L(D - L)^{-1} L^T x^m$$

$$= M^{-1} d + M^{-1} N x^m$$

$$M^{-1} = (D - L^T)^{-1} + (D - L^T)^{-1} L(D - L)^{-1}$$

$$= (D - L^T)^{-1}[(D - L) + L](D - L)^{-1}$$

$$= (D - L^T)^{-1} D(D - L)^{-1}$$

$$\text{Solve} \quad M\hat{r} = r$$

$$(D - L)D^{-1}(D - L^T)\hat{r} = r.$$

$$\text{For } w \neq 1$$

$$M_w^{-1} = (\frac{1}{w}(D - wL^T))^{-1}(\frac{2 - w}{w})D(\frac{1}{w}(D - wL))^{-1}.$$

Matlab Preconditioned Conjugate Gradient with SSOR (cgssor.m)

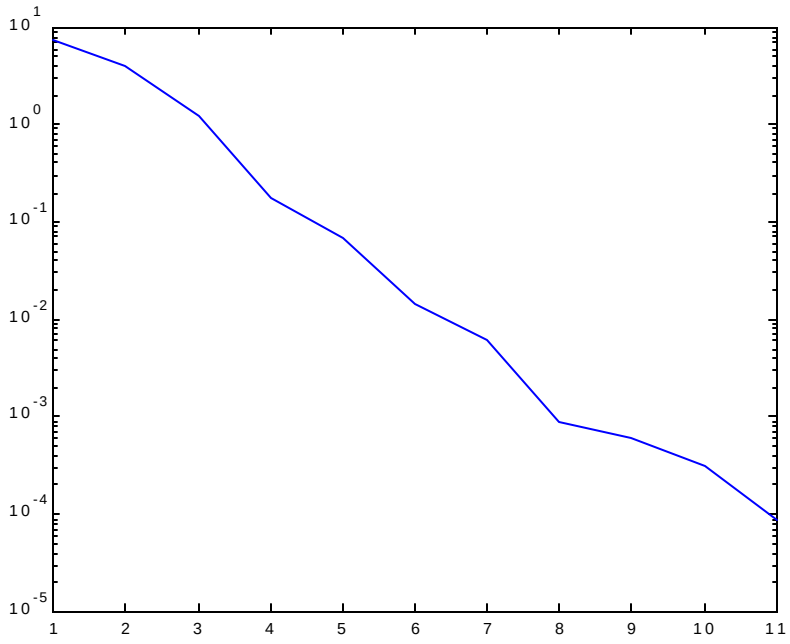
```
clear;
%
%  Solves  -uxx -uyy = 200+200sin(pi x)sin(pi y) with zero BCs
%  Uses PCG with SSOR preconditioner
%  Uses 2D arrays for the column vectors
%  Does not explicity store the matrix
%
w = 1.5;
n = 20;
h = 1./n;
u(1:n+1,1:n+1)= 0.0;
r(1:n+1,1:n+1)= 0.0;
```

```
rhat(1:n+1,1:n+1) = 0.0;
%  Define right side of PDE
for j= 2:n
   for i = 2:n
      r(i,j)= h*h*(200+200*sin(pi*(i-1)*h)*sin(pi*(j-1)*h));
   end
end



p(1:n+1,1:n+1)= 0.0;
q(1:n+1,1:n+1)= 0.0;
err = 1.0;
m = 0;
rho = 0.0;
%  Begin PCG iterations
while ((err>.0001)*(m<200))
   m = m+1;
   oldrho = rho;
%  Execute SSOR preconditioner
   for j= 2:n
      for i = 2:n
         rhat(i,j)=w*(r(i,j)+rhat(i-1,j)+rhat(i,j-1))/4.;
      end
   end
   rhat(2:n,2:n) = ((2.-w)/w)*(4.)*rhat(2:n,2:n);
   for j= n:-1:2
      for i = n:-1:2
         rhat(i,j)=w*(rhat(i,j)+rhat(i+1,j)+rhat(i,j+1))/4.;
      end
   end
%  Find conjugate direction
   rho = sum(sum(r(2:n,2:n).*rhat(2:n,2:n)));
   if (m==1)
      p = rhat;
   else
      p = rhat + (rho/oldrho)*p;
   end
%
%  Use the following line for steepest descent method
%   p=r;
%
%  Executes the matrix product q = Ap without storage of A
   for j= 2:n
      for i = 2:n
         q(i,j)=4.*p(i,j)-p(i-1,j)-p(i,j-1)-p(i+1,j)-p(i,j+1);
      end
   end
%  Executes the steepest descent segment
   alpha = rho/sum(sum(p.*q));
   u = u + alpha*p;
   r = r - alpha*q;
%  Test for convergence via the infinity norm of the residual
```

```
   err = max(max(abs(r(2:n,2:n))));
   reserr(m) = err;
end
m
semilogy(reserr)
```



Log(norm(r)) versus m for PCG with SSOR

6.5 Generalized Minimum Residual

If A is not SPD, then the PCG will not be applicable because it is based on the equivalent minimization of J(x). Two alternatives, amoung others, are

1. replace Ax = d with the normal equations $A^T Ax = A^T d$,
2. replace minimization of J(x) with the minimization of

$r(x)^T r(x)$ where $r(x) = d - Ax$.

The normal equation approach can be computationally expensive or ill-conditioned. In order to make the minimization of the residual less computationally less expensive, the minimization is done over an m dimensional subspace.

Definition. $K_m = \{x \mid x = \sum_0^{m-1} a_i A^i r_o \}$ is called a **Krylov** space. The $A^i r_0$ are called **Krylov vectors**. A slight abuse of notation is to form a matrix, also written as K_m, as $K_m = [r_0 \;\; Ar_0 \;\; \ldots \; A^{m-1} r_0 \;]$.

Definition. The **generalized residual method** is given by

$$x^m = x^0 + \sum_0^{m-1} a_i A^i r_o \quad \text{where}$$

$$r(x^m)^T r(x^m) = \min r(x)^T r(x) \text{ with } x \in x^0 + K_m.$$

If after m steps the method is restarted with x^0 replaced by x^m, then it is called the **GMRES(m)** method.

The main benefit of using the Krylov subspaces is that

AK_m is contained in K_{m+1}.

This is very useful in the solution of the minimization, which is related to finding the least squares solution of

$$A(x^0 + \sum_0^{m-1} a_i A^i r_0) = d$$

$$\sum_0^{m-1} a_i A^{i+1} r_0 = d - Ax^0 = r_0$$

$$[Ar_0 \cdots A^m r_0] \begin{bmatrix} a_0 \\ \vdots \\ a_{m-1} \end{bmatrix} = r_0$$

$$AK_m a = r_0.$$

In order to efficiently solve this least squares problem, we will construct an orthonormal basis, one column vector per iteration, of K_m. Let $V_m = [v_1 \ldots v_m]$ be such a basis. Since AK_m is contained in K_{m+1}, each column in AV_m should be a linear combination of columns in V_{m+1}.

$Av_1 = v_1 h_{11} + v_2 h_{21}$ where, by the orthonormal basis property,

$v_1^T Av_1 = h_{11}$ and $v_2^T Av_1 = h_{21}$.

$Av_2 = v_1 h_{12} + v_2 h_{22} + v_3 h_{32}$ where, by the orthonormal basis property,

$v_1^T Av_2 = h_{12}$, $v_2^T Av_2 = h_{22}$ and $v_3^T Av_2 = h_{32}$.

The matrix form of this is

$$\begin{bmatrix} Av_1 & Av_2 & \cdots \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 & \cdots \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & \cdots \\ h_{21} & h_{22} & \cdots \\ 0 & h_{32} & \cdots \\ 0 & 0 & \cdots \end{bmatrix}$$

$$AV_m = V_{m+1} H.$$

A is nxn, V_m is nxm, and H is and (m+1)xm Hessenberg matrix. The QR factorization of Hessenberg matrices are easy to compute via the Givens transformation.

The first column in V_m will be the normalized r_0

$r_0 = b\ v_1$ where $v_1^T v_1 = 1$ so that $b = (r_0^T r_0)^{1/2}$.

Hence, r_0 is the first column of V_{m+1} times b, that is,

$r_0 = V_{m+1} e_1\ b$ where $e_1 = [1\ 0.....]^T$.

Then the least squares problem can be written as

$AK_m\ \alpha = r_{0,}$ or

$AV_m\ \alpha = V_{m+1} e_1\ b$.

Proposition 10. The least squares solution of $AK_m\ \alpha = r_0$ is given by the least squares solution of $H\alpha = e_1\ b$ where $b = (r_o^T r_o)^{1/2}$ and $AV_m = V_{m+1}H$.

Proof. $AV_m\ \alpha = V_{m+1} e_1\ b$

$V_{m+1} H\ \alpha =$

The least squares solution means $R(\alpha)^T R(\alpha)$ is a minimum where

$$R(\alpha) = V_{m+1} e_1 b - V_{m+1} H \alpha .$$

Since V_{m+1} is orthonormal,

$$R(\alpha)^T R(\alpha) = (V_{m+1} e_1 b - V_{m+1} H \alpha)^T (V_{m+1} e_1 b - V_{m+1} H \alpha)$$

$$= (e_1 b - H\alpha)^T V_{m+1}{}^T V_{m+1} (e_1 b - H\alpha)$$

$$= (e_1 b - H\alpha)^T (e_1 b - H\alpha).$$

So, this is the least squares solution of $H\alpha = e_1 b$.

In order to find the least squares solution, we must solve the normal equations via the QR factors of H. Let H = QR so that the normal equation becomes

$$H^T H\alpha = H^T e_1 b$$

$$R \alpha = Q^T e_1 b.$$

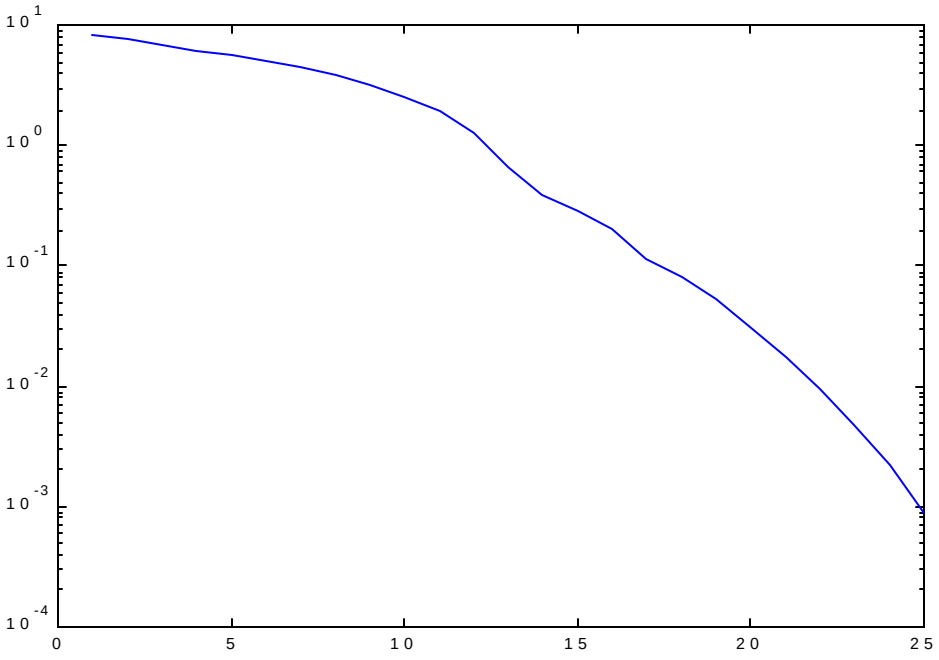
The Givens transformation can be used to construct the QR factorization of H. Moreover, the basis and Hessenberg matrix can be constructed one column per iteration. The following implementation solve the Poisson problem where the matrix product step is a sparse matrix product, and the unknowns are listed in a 2D space grid array.

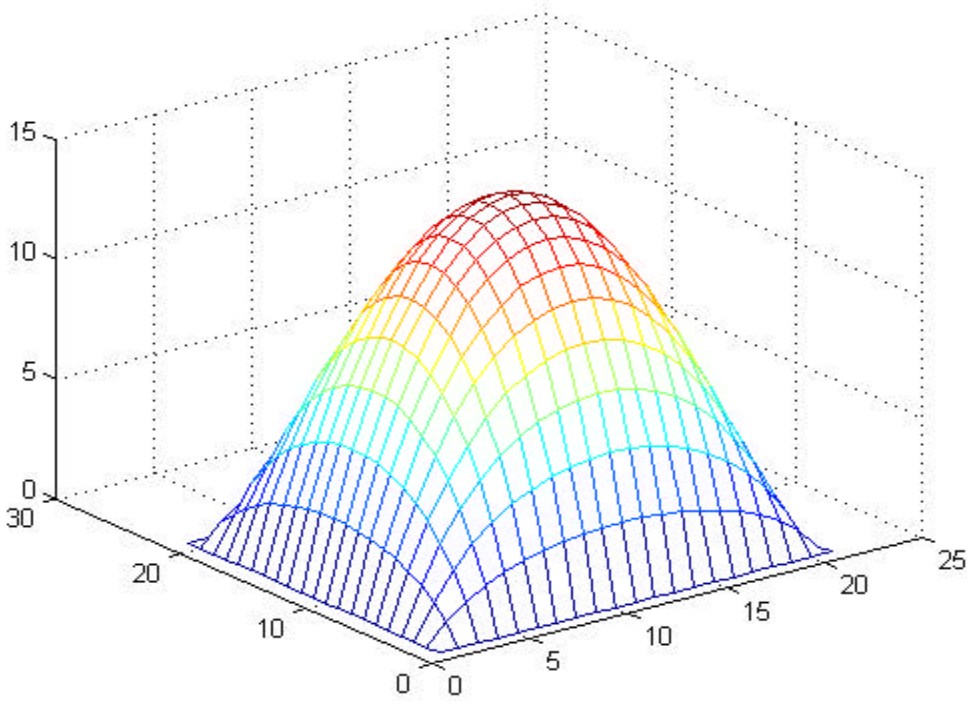
Matlab Code GMRES2d.m

```
% gmres method for Poisson equation
% see C. T. Kelley's text
% see Matlab file gmres.m
clear;
%  Input data.
nx = 20;
```

```
ny = nx;
errtol=.0001;
kmax = 30;
%  Initial guess.
x0(1:nx+1,1:ny+1) = 0.0;
x = x0;
h = zeros(kmax);
v = zeros(nx+1,ny+1,kmax);
c = zeros(kmax+1,1);
s = zeros(kmax+1,1);
b(1:nx+1,1:ny+1) = 200./(nx*nx);
r = b;
rho = sum(sum(r(2:nx,2:ny).*r(2:nx,2:ny)))^.5;
g = rho*eye(kmax+1,1);
errtol = errtol*rho;
v(2:nx,2:ny,1) = r(2:nx,2:ny)/rho;
k = 0;
%  Begin gmres loop.
while((rho > errtol) & (k < kmax))
    k = k+1;
%  Matrix vector product.
      v(2:nx,2:ny,k+1) = -v(1:nx-1,2:ny,k)-v(3:nx+1,2:ny,k)-
          v(2:nx,1:ny-1,k)-v(2:nx,3:ny+1,k)+4.*v(2:nx,2:ny,k);
%  Begin modified GS. May need to reorthogonalize.
    for j=1:k
        h(j,k) = sum(sum(v(2:nx,2:ny,j).*v(2:nx,2:ny,k+1)));
        v(2:nx,2:ny,k+1) = v(2:nx,2:ny,k+1)-h(j,k)*v(2:nx,2:ny,j);
    end
    h(k+1,k) = sum(sum(v(2:nx,2:ny,k+1).*v(2:nx,2:ny,k+1)))^.5;
    if(h(k+1,k) ~= 0)
         v(2:nx,2:ny,k+1) = v(2:nx,2:ny,k+1)/h(k+1,k);
    end
%  Apply old Givens rotations to h(1:k,k).
    if k>1
       for i=1:k-1
         hik    = c(i)*h(i,k)-s(i)*h(i+1,k);
         hipk   = s(i)*h(i,k)+c(i)*h(i+1,k);
         h(i,k) = hik;
         h(i+1,k) = hipk;
       end
    end
    nu = norm(h(k:k+1,k));
%  May need better Givens implementation.
%  Define and Apply new Givens rotations to h(k:k+1,k).
    if nu~=0
        c(k) = h(k,k)/nu;
        s(k) = -h(k+1,k)/nu;
        h(k,k) = c(k)*h(k,k)-s(k)*h(k+1,k);
        h(k+1,k) = 0;
        gk   = c(k)*g(k) -s(k)*g(k+1);
        gkp  = s(k)*g(k) +c(k)*g(k+1);
        g(k) = gk;
        g(k+1) = gkp;
    end
```

```
    rho=abs(g(k+1));
    mag(k) = rho;
 end
%  End of gmres loop.
%  h(1:k,1:k) is upper triangular matrix in QR.
  y=h(1:k,1:k)\g(1:k);
%  Form linear combination.
for i=1:k
   x(2:nx,2:ny) = x(2:nx,2:ny) + v(2:nx,2:ny,i)*y(i);
end
semilogy(mag)
%  mesh(x)
```




15
10
5
0
30
20
10
0
0
5
10
15
20
25