Day 24: 8.3, 8.4

Convergence: For $A = M - N$, iteration is $Mx^{(k+1)} = Nx^{(k)} + b$.
This covers all The standard ones.
Converges iff $\rho(b) < 1$, $\rho(b) = \rho(M^{-1}N)$ is abs. values of largest eigenvalue.
each iteration, error approx multi. by $\rho(b)$, so slow if $\rho(b) \approx 1$.

$R_{\infty}(b) = -\ln(\rho(b))$ a asymptotic convergence rate.
$\frac{1}{R_{\infty}(c)}$ approx equal to # of iterations to reduce error by factor of e.

EX: For “model problem” Poisson eqn, SOR w/ optimal $\omega$ has
$\rho(b) = 1 - 2\pi h + O(h^2)$  
$R_{\infty}(b) = 2\pi h + O(h^2)$

So smaller mesh size $h$ means slower convergence.

Q1: If $h = \frac{1}{1000\pi}$, how many iterations to reduce error by factor of e?  
$R_{\infty}(b) \approx 2\pi h/1000\pi = \frac{1}{500}$.  
$\frac{1}{R_{\infty}(c)} \approx 500$, so 500.

Q2: If we halve mesh size, how do we affect # of iterations till convergence?  
Halved $R_{\infty}(c)$, so double iterations.

Q3: Using SOR w/ optimal $\omega$, find the total computational cost for solving  
the model problem based on $m = \frac{1}{h}$, the # of grid pts in each direction.  
(remember: $m^2$ eqns. Since only 5 unknowns per eqn, updating a individual  
entry is $O(1)$. Find work per iteration and total number of iterations.)

One iteration updates $m^2$ entries, so $O(m^2)$ for 1 iteration
Need double iterations if double $m$, by last problem, so $O(m)$ iterations.
So total $O(m^3)$ flops to solve. (vs $O(m^4)$ for banded Cholesky)

Would be better if we had a method to iterate so that the number of iterations  
didn’t increase when we reduce $h$. Then would be $O(m^2)$ overall.

Multigrid methods: not in book, want go into detail, just a vague idea.  
idea: coarse grids are easy to solve. So use coarse grid soln to help  
find soln to fine grid.

This layered approach converges very rapidly, though obviously tricky to program.

8.4 descent methods.

Idea: Think of the soln of $Ax = b$ being the bottom of a bowl.
Descent methods try to find the bottom of the bowl by going down  
step by step.
Define bowl: For a s.p.d., \( J(y) = \frac{1}{2} y^T A y - y^T b \).

If \( y \) is a 2 dim vector \([x,y]\), and we think of \( J(y) \) as a height, this is exactly a paraboloid (since \( A \) is s.p.d).

Thm: For a s.p.d., there is exactly one \( x \) for which \( J(x) = \min_{y} J(y) \), and that is the soln. of \( A x = b \).

pf: Min. can only occur where gradient is zero, i.e., \( \nabla J = 0 \).
\[
\nabla J = \frac{1}{2} (\nabla y)^T A y + \frac{1}{2} y^T A (\nabla y) - (\nabla y)^T b
\]
\[
= \frac{1}{2} A y + \frac{1}{2} y^T A - b
\]
\[
= A y - b = 0 , \text{ so zero only if } y \text{ solves } A y = b . \checkmark
\]

\( J \) is often physical. For model problem, which can represent drum head, it is the elastic potential energy. The drum takes the shape that minimizes energy, so we need to find that shape, which is represented by \( y \).

Descent methods try to minimize \( J \).

Idea: Iterate \( x(0), x(1), \ldots \) reducing \( J \) each time \( (J(x(k+1)) \leq J(x(k))) \).

- For each iteration
  1. Choose direction
  2. Choose distance to go in that direction.

\( \mathbb{R}^2 \) version:

\[ \frac{1}{2} (\nabla x)^T (\nabla x) - (\nabla x^T b \text{ or } ((\nabla x)^T A) \text{ or } ((\nabla x)^T b) \text{ or } ((\nabla x)^T A (\nabla x) \text{ or } ((\nabla x)^T b) \text{ or } ((\nabla x)^T A (\nabla x) \text{ or } ((\nabla x)^T b) \text{ or } ((\nabla x)^T A (\nabla x) \text{ or } ((\nabla x)^T b) \text{ or } ((\nabla x)^T A (\nabla x) \text{ or } ((\nabla x)^T b) \text{ or } ((\nabla x)^T A (\nabla x) \text{ or } ((\nabla x)^T b) \text{ or } ((\nabla x)^T A (\nabla x) \text{ or } ((\nabla x)^T b) \text{ or } ((\nabla x)^T A (\nabla x) \text{ or } ((\nabla x)^T b) \text{ or } ((\nabla x)^T A (\nabla x) \text{ or } ((\nabla x)^T b) \text{ or } ((\nabla x)^T A (\nabla x) \text{ or } \]

\[ x^{(k+1)} = x^{(k)} + \alpha_{k} p^{(k)} \text{ vector representing direction} \]

One obvious distance to go is to minimize \( J \) in the chosen direction.

That is exact line search. (Show ex. in diagram.)

If not, then inexact line search.

Ex: G-S is descent where directions are the coord. directions and exact line search.

SOR is same, but inexact. (Over corrects.)

Q4: Exact is always better than inexact.

False: SOR better than G-S.

Just depends on method.
Day 25 8.4, 8.7

Descent methods: There is energy \( J(y) = \frac{1}{2} y^T A y - y^T b \).

For each iteration, pick direction and distance to go in that direction.
Try to minimize \( J(y) \).

Exact line search \( \Rightarrow \) choose distance so that \( J(y) \) is minimized in that direction.

**Thm:** For \( x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} \) obtained by exact line search,
\[
\alpha_k = \frac{p^{(k)^T} r^{(k)}}{p^{(k)^T} A p^{(k)}} = \frac{\langle p^{(k)}, b - Ax^{(k)} \rangle}{\langle p^{(k)}, A p^{(k)} \rangle}
\]

**pf:** Want to minimize \( g(\alpha) = J(x^{(k)} + \alpha p^{(k)}) \) for fixed \( x^{(k)}, p^{(k)} \). Thus, just one variable. Minimized when \( g'(\alpha) = 0 \), which is the above condition.

As we said before, exact line search may not be better, but it is simple.

For exact line search (and for finding \( p \) and checking convergence), need \( r^{(k)} = b - Ax^{(k)} \).

Better: \( x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} \) rearranges as
\[
\gamma^{(k+1)} = \gamma^{(k)} - \alpha_k A p^{(k)}
\]
Still a matrix-vector mult, but already had to do it for \( \alpha_k \), so no loss. So only \( O(n) \).

Generic descent algorithm:
\[
(r^{(0)} = b - Ax^{(0)}) \quad \text{(need first residual)}
\]

pick direction \( p^{(0)} \)
for some # of iterations, or till converged
\[
q^{(k)} = A p^{(k)}
\]

\[
\alpha_k = \frac{r^{(k)^T} q^{(k)}}{r^{(k)^T} A r^{(k)}}
\]

\[
x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}
\]

\[
\gamma^{(k+1)} = \gamma^{(k)} - \alpha_k A p^{(k)}
\]

pick next \( p \).
Check convergence.

Of course, really just store each new \( x, q, r, \) etc. over old ones.

So, only thing left is to pick a direction \( p \).

**Q1:** We can pick the direction towards the center? T/F. Why? Best is to pick direction is right to center. (sketch)

But choosing that direction is equivalent to solving the original problem any way! So, not really an iterative problem....

G-S chooses coord directions - show (make sure not horiz. ellipses or converges in 2...)

Steepest descent - choose steepest direction! Sketch.

What is steepest direction? Recall from Calc, that steepest direction (up)
\[
\nabla J(x^{(k)}) \]

So \( \nabla J(x^{(k)}) = A x^{(k)} - b = -\gamma^{(k)} \)

So pick \( p^{(k)} = r^{(k)} = -\nabla J(x^{(k)}) \) (steepest down)

Since already calculating \( r^{(k)} \), this is super easy to implement.
problem: converges slowly.

If the "bowl" is long and skinny, converges slowly.
And it is long and skinny if condition number is large, even just a bit.
For ex. for model problem, 160x160 grid, took even more iterations than Jacobis, and each iteration takes longer.

Idea for improvement: In a canyon it's pretty clear what's going on.
We can see that we're not changing much each step; just going back and forth.
Why not use my previous directions to help me pick my next direction.
I.e., cut to the chase, try to go down skinny direction that I know is there.

Q2: Why is this different than picking "the direction to the center" like we discussed earlier?

Different b/c not guessing from nothing! Using previous guesses!

Of course, that sounds touchy, but there's a lovely algorithm.

8.7: Conjugate Gradient method.

\[ \begin{align*}
    r &\leftarrow r - Ax \\
    p &\leftarrow r \\
    v &\leftarrow r^T r \\
    \text{Do till converged:} \\
    q &\leftarrow Ap \\
    \alpha &\leftarrow v / p^T q \\
    x &\leftarrow x + \alpha p \\
    r &\leftarrow r - \alpha q \\
    \beta &\leftarrow v_{\text{new}} / v \\
    p &\leftarrow r + \beta p \\
    v &\leftarrow v_{\text{new}} \\
\end{align*} \]

only real place it differs from Steepest descent. New direction is not \( r \) but \( r + \beta p \), i.e., combine a bit of the previous direction with the previous directions.

Converges very quickly. For ex. converges in 2/3 the iterations as SOR w/ optimum \( \omega \) for model problem, and without guessing optimum \( \omega \).

Fact: CG: at each step, minimizes \( J(y) \) not just over a line, but over the entire span of all the previous \( r \) (proof is 8.8).

Steepest descent — only over the current \( r \).
CG — over span of all previous, so never duplicate effort by going back and forth over canyon.

In fact, for 2x2 system, always converges in 2 steps!
8.5: Stopping conditions

Choices: 1. \( \frac{\| x^{(k+1)} - x^{(k)} \|}{\| x^{(k+1)} \|} < \varepsilon \). Simple, traditional, convenient. But no theoretical justification.

2. \( \frac{\| r^{(k+1)} \|}{\| b \|} < \varepsilon \). Finding residuals anyway, so easy.
   But, also, this implies that our algorithm was backwards stable.
   \( \varepsilon \)-backwards stable.
   But, actually, this is a stronger condition than necessary to show backwards stability, so we're wasting iterations.

3. \( \frac{\| r^{(k+1)} \|}{\| b \| \| x^{(k+1)} \|_2} < \varepsilon \) Need to estimate \( \| A \|_2 \), which is annoying, but only need to do once.
   \( \varepsilon \)-backwards stable if and only if this is true.
   So don't waste iterations, trying to get "accuracy" that means nothing.
   Sometimes this difference can be extreme, but usually, any of these is fine.

8.6: Preconditioners

Idea: Trying to minimize \( J(y) \).
   But if the bowl of \( J \) is long and skinny, iterative methods struggle.
   So, try to modify the problem to make the \( J \) more well rounded.
   Since ill-conditioned \( \Rightarrow \) long + skinny, we will precondition our problem.

Take \( Ax = b \), instead solve equivalent system \( \tilde{A} \tilde{x} = \tilde{b} \), which is simpler.

Ex: Take \( M = A \) (a splitting matrix), then
(\( M^{-1}A \))\( x = M^{-1}b \). Solve this. May be easier.

Problem is that this loses symmetry even if we had it before.

Better: If \( M \) is s.p.d., \( M = R^T R \). Instead of using \( \tilde{A} = M^{-1}A \),
use \( \tilde{A} = R^{-T} A R^{-1} \). In other words, solve the problem
\( (R^{-T} A R^{-1}) (Rx) = R^{-T} b \)
\( \tilde{A} \tilde{x} = \tilde{b} \)

So can just run algorithm using \( \tilde{A}, \tilde{x}, \tilde{b} \).

Q1: Why might this not be efficient?
   Naively, matrix mutt to find \( \tilde{A} \) is \( O(n^3) \), though we could prob do in \( O(n^2) \) since \( R \) is simple.
   Also, once we find \( \tilde{x} \), we have to solve \( Rx = \tilde{x} \), which is prob \( O(n^2) \).
   But, remember, we Kind of want \( O(n) \)!
   So, not ideal.
   Fortunately, there's a better way to arrange the work.
   Rather than solve using \( \tilde{A}, \tilde{x}, \tilde{b} \), do equivalent calculations with original quantities.
For example, the tilde version has $\tilde{x} = \tilde{x} + \alpha \tilde{p}$.

But can show $\tilde{x} = x$. (eventually ...) $X = R^{-1} \tilde{x}$, so $p = R^{-1} \tilde{p}$ should hold, since that way change to search direction is same as change to $x$.

So, then $x \leftarrow x + \alpha \tilde{p}$ is equiv to $\tilde{x} \leftarrow \tilde{x} + \alpha \tilde{p}$, if we mult. by $R^{-1}$ (on both sides).

Similarly, $\tilde{z} \leftarrow A \tilde{p} = R^T A \rho$. $R^T \tilde{q} = A \rho$, so call $q = R^T \tilde{q}$, Then have $q \leftarrow A \rho$.

Q2: Using the line $\tilde{r} \leftarrow \tilde{r} - \alpha \tilde{q}$, find the correct transformation for $\tilde{r}$.

If I want this nice, need $\tilde{r} \leftarrow R^{-T} \tilde{r}$. Then can mult. through by $R^T$ to get $r \leftarrow r - \alpha q$.

Can go step by step to get this to work. Everything seems to transform to untilded version, so what did we gain?

But line $p \leftarrow ?$ (choosing direction) is different!

For steepest descent, $p \leftarrow r$ is new direction.

So $R p \leftarrow R^T \tilde{r}$

$p \leftarrow R^{-1} R^{-T} r = M^{-1} r$. So only change is mult. by $M^{-1}$ to find $p$!

Inefficient to actually multiply by $M$, though.

Recall, for splitting $A = M - N$

If $x^{(0)} = 0$, $x^{(1)} = M^{-1} r^{(0)}$

Don't think of this as solve to $Ax = b$, rather, this says

"If we do 1 iteration of the method represented by splitting matrix $M$, with initial guess 0, to the system $Ax = r$, we find $M^{-1} r$.

So program is not $p \leftarrow M^{-1} r$.

It's: $p \leftarrow$ one iteration of method for $M$, $x^{(0)} = 0$, solving $Ax = r$.

Don't use SOR -> not symmetric. Use SSOR.

How good?

\[ \text{steepest descent, no precondition, } \approx 50,000 \text{ iterations} \]

\[ \text{SSOR, } \omega = \text{ideal } \approx 260 \]

\[ \text{Conj. grad, no precondition, } \approx 444 \]

\[ \text{SSOR, } \omega = \text{ideal } \approx 46 \]

Preconditioner does double or triple effort per iteration, but worth it.

Even better ones exist than SSOR.