Notes

- Assignment 1 due tonight
  (email me by tomorrow morning)

The Power Method

- Start with some random vector v, $\|v\|_2=1$
- Iterate $v=(Av)/\|Av\|$
- The eigenvector with largest eigenvalue tends to dominate
- How fast?
  - Linear convergence, slowed down by close eigenvalues

Shift and Invert (Rayleigh Iteration)

- Say the eigenvalue we want is approximately λ_k
- The matrix $(A-\lambda_k I)^{-1}$ has the same eigenvectors as A
- But the eigenvalues are $\mu = \dfrac{1}{\lambda - \lambda_k}$
- Use this in the power method instead
- Even better, update guess at eigenvalue each iteration: $\lambda_{k+1} = v_{k+1}^T A v_{k+1}$
- Gives cubic convergence! (triples the number of significant digits each iteration when converging)

Maximality and Orthogonality

- Unit eigenvectors v_1 of the maximum magnitude eigenvalue satisfy

$$\|Av_1\|_2 = \max_{\|u\|=1} \|Au\|_2$$

- Unit eigenvectors v_k of the k'th eigenvalue satisfy

$$\|Av_k\|_2 = \max_{\substack{\|u\|=1 \\ u^T v_i = 0, i<k}} \|Au\|_2$$

- Can pick them off one by one, or….

Orthogonal iteration

- Solve for lots (or all) of eigenvectors simultaneously
- Start with initial guess V
- For k=1, 2, …
  - Z=AV
  - VR=Z (QR decomposition: orthogonalize Z)
- Easy, but slow
  (linear convergence, nearby eigenvalues slow things down a lot)

Rayleigh-Ritz

- Aside: find a subset of the eigenpairs
  - E.g. largest k, smallest k
- Orthogonal estimate V ($n\times k$) of eigenvectors
- Simple Rayleigh estimate of eigenvalues:
  - $\mathrm{diag}(V^TAV)$
- Rayleigh-Ritz approach:
  - Solve $k\times k$ eigenproblem V^TAV
  - Use those eigenvalues (Ritz values) and the associated orthogonal combinations of columns of V
  - Note: another instance of **"assume solution lies in span of a few basis vectors, solve reduced dimension problem"**

Solving the Full Problem

- Orthogonal iteration works, but it's slow
- First speed-up: make A tridiagonal
  - Sequence of symmetric Householder reflections
  - Then Z=AV runs in $O(n^2)$ instead of $O(n^3)$
- Other ingredients:
  - Shifting: if we shift A by an exact eigenvalue, $A-\lambda I$, we get an exact eigenvector out of QR (the last column)
    - improves on linear convergence
  - Division: once an offdiagonal is almost zero, problem separates into decoupled blocks

Nonlinear optimization

- Switch gears a little:
  we've already seen plenty of instances of minimizing, with linear least-squares
- What about nonlinear problems?
- Find $x = \arg\min_x f(x)$
- f(x) is called the **objective**
- This is an **unconstrained** problem, since no limits on x.

Classes of methods

- Only evaluate f:
  - Stochastic search, pattern search, cyclic coordinate descent (Gauss-Seidel), genetic algorithms, etc.
- Also evaluate $\partial f/\partial x$ (gradient vector)
  - Steepest descent and relatives
  - Quasi-Newton methods
- Also evaluate $\partial^2 f/\partial x^2$ (Hessian matrix)
  - Newton's method and relatives

Steepest Descent

- The gradient is the direction of fastest change
  - Locally, f(x+dx) is smallest when dx is in the direction of negative gradient ∇f
- The algorithm:
  - Start with guess $x^{(0)}$
  - Until converged:
    - Find direction $d^{(k)} = -\nabla f\left(x^{(k)}\right)$
    - Choose step size $\alpha^{(k)}$
    - Next guess is $x^{(k+1)} = x^{(k)} + \alpha^{(k)} d^{(k)}$

Convergence?

- At global minimum, gradient is zero:
  - Can test if gradient is smaller than some threshold for convergence
  - Note: scaling problem: min A*f(B*x)+C
- However, gradient is also zero at
  - Every local minimum
  - Every local maximum
  - Every saddle-point

Convexity

- A function is **convex** if

$$f\left(\alpha x + (1-\alpha)y\right) \le \alpha f\left(x\right) + (1-\alpha) f\left(y\right)$$
$$\alpha \in [0,1]$$

- Eliminates possibility of multiple strict local mins
- Strictly convex: at most one local min
- Very good property for a problem to have!

Selecting a step size

- Scaling problem again:
  physical dimensions of x and gradient may not match
- Choosing a step too large:
  - May end up further from minimum
- Choosing a step too small:
  - Slow, maybe too slow to actually converge
- **Line search**: keep picking different step sizes until satisfied