Lecture 11 - Iterative Methods

* AMA / Quiz / Prayer
* midcourse evaluation

I. Direct vs. Iterative Methods

* Gauss Elimination is an example of a "direct method" for solving a linear system. With the exception of some numerical issues, it almost always works. But it is slow. We'll have more to say about what "slow" means next time.

* Iterative methods are a class of algorithms for solving linear systems that don't always work, but when they do, the can be much faster than Gauss Elimination.

Gauss Elim (direct)	Iterative
always works	don't always work
slow	fast

* Iterative methods employ a "guess and check" strategy. We'll demonstrate this with the following example:

solve: $\underline{\underline{A}}\,\underline{x} = \underline{b}$ using an iterative method.

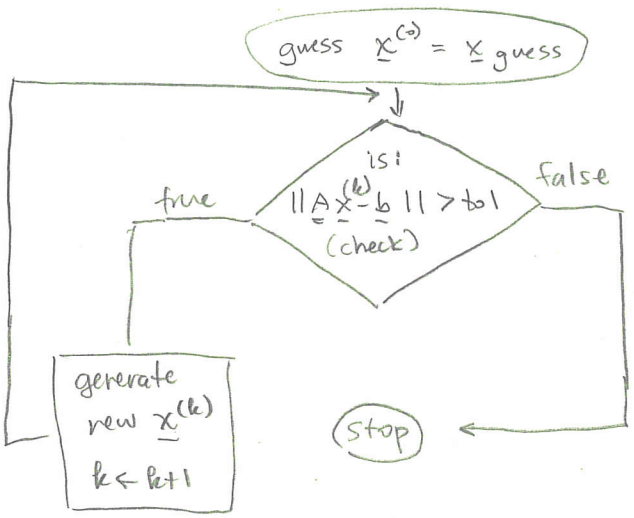
note: $\underline{\underline{A}}\,\underline{x} - \underline{b} = \underline{0}$ is a re-written form of $\underline{\underline{A}}\,\underline{x} = \underline{b}$,

$\underline{0}$ is a vector. (residual form)

$\|y\|$ is the norm of a vector

$$\|y\| = \sqrt{\sum_{i=0}^{n-1} y_i^2}$$

(like a distance formula, tells you how "big" y is)



- $\underline{x}_{guess}$ could be anything, e.g. $\underline{0}$
- tol is some small number we pick, e.g. 10^{-8}.
- $\|\underline{A}\underline{x}^{(k)} - \underline{b}\|$ is called the residual.
- Things we can choose: ① $\underline{x}_{guess}$, ② way to generate $\underline{x}^{(k)}$

Q: what is this structure? A: A while loop!

```
x = x_guess
tol = 10^-8
k = 0
res = norm(Ax-b)        ← need some function to do this.

while (res > tol):

    # generate new x

    k = k+1

    eps = norm(Ax-b)
```

* There are 3 possible results from this loop:

(1) Convergence: we reach an answer. The residual $\to 0$.

(2) "stall out": we don't reach an answer, but $\underline{x}^{(k)}$ stops changing. We could keep going, but it won't help. ($\|\underline{x}^{(k+1)} - \underline{x}^{(k)}\| \to 0$).

(3) Divergence: we don't reach an answer. $\underline{x}^{(k)}$ "blows up" and gets farther & farther away from an answer.

* For a given $\underline{\underline{A}}$ & $\underline{b}$ (a linear system) we can only change $\underline{x}$ guess or the method to get $\underline{x}^k$. Some $\underline{\underline{A}}$'s & $\underline{b}$'s won't work! Some guesses and methods won't work!

II. Jacobi's method.

* Jacobi's method is a way of gererating a new $\underline{x}$. It looks a lot like back substitution without the forward elimination.

$$a_{00}x_0 + a_{01}x_1 + \dots + a_{0,n-1}x_{n-1} = b_0$$

$$a_{10}x_0 + a_{11}x_1 + \dots + a_{1,n-1}x_{n-1} = b_1$$

$$\vdots$$

$$a_{n-1,0}x_0 + a_{n-1,1}x_n + \dots + a_{n-1,a_{n-1}}x_{n-1} = b_{n-1}$$

↓ Solve for x_0 in first line,
x_1 in the second line, ...
x_{n-1} in the last line:

$$x_0 = \frac{1}{a_{0,0}}\left[b_0 - a_{01}x_1 - \dots - a_{0,n-1}x_{n-1}\right]$$

$$x_1 = \frac{1}{a_{1,1}}\left[b_1 - a_{10}x_0 - \underbrace{a_{1,2}x_2} - \dots - a_{1,n-1}x_{n-1}\right]$$

skip x_1

$$\vdots$$

$$x_{n-1} = \frac{1}{a_{n-1,a_{n-1}}}\left[b_{n-1} - a_{n-1,0}x_0 - a_{n-1,1}x_1 - \dots - a_{n-1,n-2}x_{n-2}\right]$$

* We haven't really solved for $\underline{x}$ on the LHS. We need to know it on the RHS too! (In Back substitution, we worked with a special case where we did know the RHS).

* So, we put $x^{(k)}$ (or the guess) on the RHS and we get out $x^{(k+1)}$ on the LHS.

* We can write the above formula more compactly:

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left[b_i - \sum_{\substack{j=0 \\ j \neq i}}^{n-1} a_{i,j} x_j^{(k)} \right]$$

Activity: Code Jacobi's method for the students in class.

```python
#-------------------------------------------------------------
# Lecture 11 -- In-class activity Jacobi's method
#-------------------------------------------------------------

import numpy as np

# Solve the linear system, A.x = b
#
# where A =
# [  2  1 ]
# [  1 -2 ]
#
# and b =
# [   2  ]
# [  -2 ]
#
# Using Jacobi's (iterative) method

# One that blows up?
# A =
# [  2  3 ]
# [  4 -3 ]
#
# and b =
# [   2  ]
# [  -2 ]

# Arrays in problem statement
A = np.array([[ 2,  1],
              [ 1, -2]])

b = np.array([2, -2])

n = len(b)
x = np.zeros(n)

# matrix-vector product
def dot(M, v):
    out = np.zeros(n)
    for i in range(n):
        for j in range(n):
            out[i] += M[i,j]*v[j]
    return out

# norm of a vector
def norm(v):
    out = 0
    for i in range(n):
        out += v[i]**2
    return np.sqrt(out)

# Iteration loop

tol = 1e-8
x_guess = np.zeros(n)
k=0
k_max = 100

x = x_guess
```

```
res = norm(dot(A, x)-b)
print(k, res)

while(res > tol):

    ### beginning of jacobi's method for updating x ###
    x_old = np.copy(x)

    # first element
    i=0
    S = b[i]
    for j in range(1,n):
        S -= A[i, j]*x_old[j]
    x[i] = S/A[i, i]

    # 2nd through (n-1)th element
    for i in range(1, n-1):

        S = b[i]
        # before i == j
        for j in range(0, i):
            S -= A[i, j]*x_old[j]

        # after i == j
        for j in range(i+1, n):
            S -= A[i, j]*x_old[j]

        x[i] = S/A[i,i]

    # last element
    i=n-1
    S = b[i]
    for j in range(0, n-1):
        S -= A[i, j]*x_old[j]

    x[i] = S/A[i,i]
    ### end of Jacobi's method for updating x ###

    k += 1 # update counter
    res = norm(dot(A, x) - b) # calculate residual
    print(k, res) # print to screen so we can see our progress

    if (k >= k_max):
        break

print('\nsolution: ', x)
print('check the solution: A.x = ', dot(A, x))
```