Course Notes: Week 2.

Math 270C: Applied Numerical Linear Algebra

1 Lecture 4: Givens QR and GMRES (4/4/11)

At each step in the Krylov method, we must solve a least squares problem to find the best $\mathbf{x}^k$ in the k^{th} Krylov space. For the record, this Krylov space method is called GMRES and was invented in 1986. Notably, this is after the conjugate gradient method (CG). CG can be viewed as a the analogous procedure, but we will get to that detail later. For now, it remains to show that we can solve the least squares problem for $\mathbf{x}^k$ efficiently. We will see that the QR solution to the least squares problem is possible with one Givens rotation at each time step because of the fortuitous properties of the Arnoldi iteration. Recall that the Arnoldi iterations yield the following recursive relationship between the search directions

$$\mathbf{A}\mathbf{q}_k = \sum_{i=1}^{k+1} h_{ik}\mathbf{q}_i, \quad k = 1, 2, ..., n-1$$

In other words, $\mathbf{A}\mathbf{Q}^k = \mathbf{Q}^{k+1}\mathbf{H}^k$ for the upper Hessenberg $\mathbf{H}^k \in \mathbb{R}^{(k+1)\times k}$

$$\mathbf{A} = \begin{pmatrix} h_{11} & h_{12} & \dots & & h_{1k} \\ h_{21} & h_{22} & \dots & & \vdots \\ 0 & h_{32} & & & \\ \vdots & 0 & \ddots & \ddots & \ddots \\ 0 & ... & & 0 & 0 & h_{k+1k} \end{pmatrix}$$

The upper Hessenberg structure will enable an efficient QR procedure that can be computed at $O(k)$ cost at the k^{th} iteration.

1.1 QR and Least Squares

I will quickly talk about the use of the QR factorization in solving the least squares problem. This is 270B material, but I add it here for quick reference. We can describe the general overdetermined least squares problem as:

$$\mathbf{x} = \underset{\mathbf{y}}{\text{argmin}} \; ||\mathbf{b} - \mathbf{A}\mathbf{y}||_2$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$ where $m \geq n$. You can solve this in a few different ways, but one very popular method is via QR factorization. A QR factorization of the matrix $\mathbf{A}$ yields

$$\mathbf{A} = \mathbf{Q}\hat{\mathbf{R}}, \quad \text{where} \quad \mathbf{Q}\mathbf{Q}^T = \mathbf{I} \quad \text{and} \quad \hat{\mathbf{R}} \quad \text{is upper triangular.}$$

Specifically, $\mathbf{Q} = [\hat{\mathbf{q}}_\mathbf{1}, \hat{\mathbf{q}}_\mathbf{2}, \dots, \hat{\mathbf{q}}_\mathbf{m}] \in \mathbb{R}^{m\times m}$ and $\hat{\mathbf{R}} \in \mathbb{R}^{m\times n}$ with

$$\hat{\mathbf{R}} = \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix}, \quad \mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ 0 & r_{22} & \dots & r_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & \dots & & r_{nn} \end{pmatrix} \in \mathbb{R}^{n\times n}$$

Because the 2-norm of a vector is invariant under orthogonal transformations, we can write the solution $\mathbf{x}$ equivalently as

$$\mathbf{x} = \underset{\mathbf{y}}{\text{argmin}} \; ||\mathbf{Q}^T (\mathbf{b} - \mathbf{A}\mathbf{y})||_2 = \underset{\mathbf{y}}{\text{argmin}} \; ||\mathbf{Q}^T\mathbf{b} - \hat{\mathbf{R}}\mathbf{y}||_2$$

and

$$\underset{\mathbf{y}}{\text{argmin}} \; ||\mathbf{Q}^T\mathbf{b} - \hat{\mathbf{R}}\mathbf{y}||_2 = \sqrt{||\begin{pmatrix} \hat{\mathbf{q}}_{n+1}^T\mathbf{b} \\ \hat{\mathbf{q}}_{n+2}^T\mathbf{b} \\ \vdots \\ \hat{\mathbf{q}}_m^T\mathbf{b} \end{pmatrix}||_2^2 + ||\begin{pmatrix} \hat{\mathbf{q}}_1^T\mathbf{b} \\ \hat{\mathbf{q}}_2^T\mathbf{b} \\ \vdots \\ \hat{\mathbf{q}}_n^T\mathbf{b} \end{pmatrix} - \mathbf{R}\mathbf{y}||_2^2}$$

Therefore if we call

$$\hat{\mathbf{b}} = \begin{pmatrix} \hat{\mathbf{q}}_{n+1}^T\mathbf{b} \\ \hat{\mathbf{q}}_{n+2}^T\mathbf{b} \\ \vdots \\ \hat{\mathbf{q}}_m^T\mathbf{b} \end{pmatrix}$$

then the least squares solution $\mathbf{x}$ is equivalently expressed as

$$\mathbf{x} = \underset{\mathbf{y}}{\text{argmin}} \; ||\hat{\mathbf{b}} - \mathbf{R}\mathbf{y}||_2$$

and of course $||\hat{\mathbf{b}} - \mathbf{R}\mathbf{y}||_2 = 0$ when $\mathbf{y} = \mathbf{R}^{-1}\hat{\mathbf{b}}$ and since $\mathbf{R}$ is upper triangular, this is trivial to compute with back substitution. Of course, computing the QR factorization of a given matrix is very expensive in general. It is remarkable that the Arnoldi iteration will let us update our QR factorization from the previous iteration with very minimal cost.

1.1.1 QR with Arnoldi

The GMRES iterate $\mathbf{x}^k$ is the solution of the least squares problem

$$\mathbf{x}^k = \mathbf{Q}^k\lambda^k, \quad \text{where} \quad \lambda^k = \underset{\lambda}{\text{argmin}} \; ||\mathbf{b} - \mathbf{A}\mathbf{Q}^k\lambda||_2.$$

Using the properties of the Arnoldi iteration

$$\lambda = \underset{\lambda}{\text{argmin}} \; ||\mathbf{b} - \mathbf{Q}^{k+1}\mathbf{H}^k\lambda||_2.$$

Now, $\mathbf{Q}^n = \left[\mathbf{Q}^{k+1}, \mathbf{q}^{k+2}, \mathbf{q}^{k+3}, \ldots, \mathbf{q}^n\right] \in \mathbb{R}^{n\times n}$. And again because the 2-norm of a vector is invariant under orthogonal transformations,

$$||\mathbf{b} - \mathbf{Q}^{k+1}\mathbf{H}^k\lambda||_2 = ||\left(\mathbf{Q}^n\right)^T\left(\mathbf{b} - \mathbf{Q}^{k+1}\mathbf{H}^k\lambda\right)||_2.$$

Now,

$$\left(\mathbf{Q}^n\right)^T\mathbf{b} = \begin{pmatrix} \mathbf{q}_1^T\mathbf{b} \\ \mathbf{q}_2^T\mathbf{b} \\ \vdots \\ \mathbf{q}_n^T\mathbf{b} \end{pmatrix} \quad \text{and} \quad \left(\mathbf{Q}^n\right)^T\mathbf{Q}^{k+1} = \begin{pmatrix} \mathbf{I}_{k+1} \\ \mathbf{0} \end{pmatrix} \in \mathbb{R}^{n\times(k+1)}$$

where $\mathbf{I}_{k+1} \in \mathbb{R}^{(k+1)\times(k+1)}$ is the $(k+1) \times (k+1)$ identity. Therefore,

$$|| \left(\mathbf{Q}^n\right)^T \left(\mathbf{b} - \mathbf{Q}^{k+1}\mathbf{H}^k\lambda\right) ||_2 = \sqrt{|| \begin{pmatrix} \mathbf{q}_{k+2}^T\mathbf{b} \\ \mathbf{q}_{k+3}^T\mathbf{b} \\ \vdots \\ \mathbf{q}_n^T\mathbf{b} \end{pmatrix} ||_2^2 + || \begin{pmatrix} \mathbf{q}_1^T\mathbf{b} \\ \mathbf{q}_2^T\mathbf{b} \\ \vdots \\ \mathbf{q}_{k+1}^T\mathbf{b} \end{pmatrix} - \mathbf{H}^k\lambda||_2^2}$$

and we can see then that λ^k is the solution of the easier least squares problem

$$\lambda^k = \underset{\lambda}{\text{argmin}} \; || \begin{pmatrix} \mathbf{q}_1^T\mathbf{b} \\ \mathbf{q}_2^T\mathbf{b} \\ \vdots \\ \mathbf{q}_{k+1}^T\mathbf{b} \end{pmatrix} - \mathbf{H}^k\lambda||_2.$$

This is easier than the original least squares problem for λ^k for a number of reasons. It is important to note them here because this is one of the main points of the Arnoldi iteration. The least squares problem involving $\mathbf{H}^k$ above is easier because:

- It is a $(k+1) \times k$ dimensional problem rather than $n \times k$ dimensional and we hope that in practice $k << n$.
- $\mathbf{H}^k$ is upper Hessenberg and thus nearly upper triangular as is. That is, it is very likely that Householder or Givens type transforms can efficient compute the QR factorization of $\mathbf{H}^k$.
- $\mathbf{H}^{k+1} = \begin{pmatrix} & h_{1k+1} \\ \mathbf{H}^k & h_{2k+1} \\ & \vdots \\ & h_{k+1k+1} \\ \mathbf{0} & h_{k+2k+1} \end{pmatrix} \in \mathbb{R}^{(k+2)\times(k+1)}$. In other words, $\mathbf{H}^{k+1}$ is exactly the same as $\mathbf{H}^k$ except for one the additional column $\begin{pmatrix} h_{1k+1} \\ h_{2k+1} \\ \vdots \\ h_{k+2k+1} \end{pmatrix}$ so we can expect that there QR factorizations are similar.

1.2 Givens QR

A Givens rotation is an efficient way to introduce zeros by multiplying with a low rank orthogonal matrix. For example, this is used to introduce zeros in $\mathbf{A}$ in the process of transforming it into $\mathbf{R}$ in the QR factorization. The Givens rotation matrix $\mathbf{G}(i,k,\theta) \in \mathbb{R}^{n\times n}$ is given by the following

formula

$$\mathbf{G}(i,k,\theta) = \begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ i \\ \vdots \\ k \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & \dots & i & \dots & k & \dots & n \end{array} \\ \begin{pmatrix} 1 & 0 & \dots & 0 & \dots & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & & \ddots & & & & & \\ 0 & & & c & & s & & 0 \\ & & & & & & & \\ 0 & & & -s & & c & & 0 \\ & & & & & & & \\ 0 & \dots & & & & & 0 & 1 \end{pmatrix} \end{array}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. This matrix has the following action on a vector $\mathbf{x}$: $\mathbf{y} = \mathbf{G}(i,k,\theta)^T\mathbf{x}$, $y_j = \begin{cases} cx_i - sx_k, & j = i \\ sx_i + cx_k, & j = k \\ x_j, & j \neq i,k \end{cases}$ In other words, the rotation can be chosen ($c = \frac{x_i}{\sqrt{x_i^2+x_k^2}}$, $s = \frac{-x_k}{x_i^2+x_k^2}$) so that $y_k = 0$. For example in two dimensions, this would be rotation of the vector $\mathbf{x}$ so that it lies on completely on the x axis (i.e. no y component). Note that this is a rotation and the norm of $\mathbf{x}$ is preserved.

1.3 Givens QR for $\mathbf{H}^{k+1}$

I will give an example in the case of computing the QR factorization of $\mathbf{H}^3$ from the QR factorization of $\mathbf{H}^2$. The general case is similar.

$$\mathbf{H}^2 = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \\ 0 & h_{32} \end{pmatrix}, \quad \mathbf{H}^3 = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & h_{32} & h_{33} \\ 0 & 0 & h_{43} \end{pmatrix}$$

It is easy to see that two Givens rotations will yield the QR factorization of $\mathbf{H}^2$:

$$\mathbf{G}(1,2,\theta_1)^T\mathbf{H}^2 = \begin{pmatrix} X & X \\ 0 & X \\ 0 & h_{32} \end{pmatrix}, \quad \mathbf{G}(2,3,\theta_2)^T\mathbf{G}(1,2,\theta_1)^T\mathbf{H}^2 = \begin{pmatrix} X & X \\ 0 & X \\ 0 & 0 \end{pmatrix}$$

therefore we only need one more Givens rotation to get the QR factorization of $\mathbf{H}^3$:

$$\mathbf{G}(3,4,\theta_3)^T\mathbf{G}(2,3,\theta_2)^T\mathbf{G}(1,2,\theta_1)^T\mathbf{H}^3 = \begin{pmatrix} X & X & X \\ 0 & X & X \\ 0 & 0 & X \\ 0 & 0 & 0 \end{pmatrix}$$

Thus, at the k^{th} iteration we only need to apply $\mathbf{G}(1,2,\theta_1)^T, \mathbf{G}(2,3,\theta_2)^T, \dots, \mathbf{G}(k-1,k,\theta_k-1)^T$ to the last column in $\mathbf{H}^k$: $\begin{pmatrix} h_{1k} \\ h_{2k} \\ \vdots \\ h_{k+1k} \end{pmatrix}$ and then determine and apply $\mathbf{G}(k,k+1,\theta_k)^T$ to the result because $\mathbf{G}(k,k+1,\theta_k)^T$ will have no effect on the previous columns because of the upper triangular/Hessenberg structure. Therefore, it is $O(k)$ cost to update the QR factorization of $\mathbf{H}^k$ given the work done at the previous iteration.

2 Lecture 5: GMRES Cost, Storage and Convergence Behavior (4/8/11)

The GMRES algorithm requires a relatively high amount of storage and becomes very costly when the number of iterations is very high. We previously discussed the dependence of the number of iterations needed on the behavior of the eigenvalues of the matrix. We will generalize that result in this lecture. However first it will be good to summarize the GMRES algorithm in terms of storage cost and flops at the k^{th} iteration. In the process, we will write the psuedocode for the algorithm. This is where you really must think carefully about each step in the algorithm to make sure we are as efficient as possible. The process is complicated enough on paper let alone in the code so we must think carefully about each step.

2.1 Storage

The most obvious storage requirements are $\mathbf{b}, \mathbf{x} \in \mathbb{R}^n$ and $\mathbf{A} \in \mathbb{R}^{n\times n}$. If $\mathbf{A}$ is sparse then this is basically $O(n)$ floats or doubles (specifically, if $\mathbf{A}$ has at most l non-zero entries per row, then we would need to store $(l+2)n$ floats or doubles to set up the problem). The algorithm GMRES must store all previous search directions (i.e. all of the $\mathbf{q}_i$). Each $\mathbf{q}_i$ is in $\mathbb{R}^n$ and so if we are prepared to take m iterations then we have to store nm additional floats or doubles. Also, we need to store a number of things associated with the least squares problems we have to solve at each step. When we solve the least squares problem, we get $\lambda^k \in \mathbb{R}^k$ where the k^{th} iteration $\mathbf{x}^k = \mathbf{Q}^k\lambda^k$. Assuming we are prepared to take m iterations, we will need m floats or doubles for λ^k. We will also need to store *something* related to the upper Hessenberg $\mathbf{H}^k$. In practice, we will only need the upper triangularized version of the $\mathbf{H}^k$: $\mathbf{R}^k \in \mathbb{R}^{k\times k}$. $\mathbf{R}^k \in \mathbb{R}^{k\times k}$ will require $\frac{k^2-k}{2}$ floats or doubles at the k^{th} iteration so if we are prepared to take m iterations then we will need to store $\frac{m^2-m}{2}$ floats or doubles. An extra storage cost at the k^{th} iteration is the rightmost column in $\mathbf{H}^k$ that needs to be upper-triangularized: $\mathbf{h}^k = \begin{pmatrix} h_{1k} \\ h_{2k} \\ \vdots \\ h_{k+1k} \end{pmatrix}$. This will be at most m floats or doubles more. Also, we will need the rotated version of $\mathbf{b}$ needed in the least squares problems, but we can actually overwrite the original $\mathbf{b}$ with this information so we can neglect that. Similarly, there is no need to store $\mathbf{x}^k$ because we can overwrite $\mathbf{x}$ with that. Also, we need the Givens rotations $\mathbf{G}(k, k+1, \theta_k)$. These can be defined from the two floats (or doubles) c_k and s_k (the k and $k+1$ can be inferred from the location of the $\mathbf{G}(k, k+1, \theta_k)$ in the array storing them). I.e. we need $2m$ floats or doubles for the Givens rotations. Therefore, summarizing the storage costs (assuming a sparse $\mathbf{A}$) we need $(l+m+2)n + \frac{m^2-m}{2} + 4m$ floats or doubles. Thus, if we are running very large scale problems where n is very large, we would not want m to be much larger than l in practical application (at least as far as storage is concerned, the computational complexity also grows as we take more iterations).

2.2 Practical steps of the algorithm

There are some interesting observations you can make to save some computation with GMRES. The biggest one is that h_{k+1k} is equal to the residual at the k^{th} iteration. This is useful because it allows you to avoid forming $\mathbf{x}^k = \mathbf{Q}^k\lambda^k$ explicitly to check if you have finished. This is important because that calculation is $O(nk)$ flops and will be increasingly expensive. WIth this (and a few more observations you can sort out below), my suggested pseudocode for GMRES is:

$\mathbf{q}_1 \leftarrow \frac{\mathbf{b}}{||\mathbf{b}||_2}, h_{10} \leftarrow ||\mathbf{b}||_2, \mathbf{x}^1 \leftarrow \mathbf{0}, \lambda^1 \leftarrow 0, \mathbf{b} \leftarrow \begin{pmatrix} ||\mathbf{b}||_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$;

for $k = 1$ to m **do**
  if residual $< tol$, return $\mathbf{x} = \mathbf{Q}^k \lambda^k$; $//O(nk)$ flops
  $\mathbf{q}_{k+1} \leftarrow \mathbf{A}\mathbf{q}_k$; $//O(ln)$ flops
  for $i = 1$ to k **do**
    $h_{ik} \leftarrow \mathbf{q}_i^T \mathbf{q}_{k+1}$; // Entries in the rightmost column for the QR ($O(n)$ flops)
    $\mathbf{q}_{k+1} \leftarrow \mathbf{q}_{k+1} - h_{ik}\mathbf{q}_i$; // Graham Schmidt the search direction, ($O(n)$ flops)
  end for
  $h_{k+1k} \leftarrow ||\mathbf{q}_{k+1}||_2$;// $O(n)$ flops and one square root
  $\mathbf{q}_{k+1} \leftarrow \frac{\mathbf{q}_{k+1}}{h_{k+1k}}$; // $O(n)$ flops

  // Now solve the least squares problem: $\lambda^k = \text{argmin } || \begin{pmatrix} ||\mathbf{b}||_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix} - \mathbf{H}^k \lambda||_2$

  for $i = 1$ to k-1 **do**
    $\mathbf{h}^k \leftarrow \mathbf{G}(i, i+1, \theta_i)\mathbf{h}^k$;// 6 flops
  end for
  Compute $\mathbf{G}(k, k+1, \theta_k)$;// 5 flops and one square root
  $\mathbf{b} \leftarrow \mathbf{G}(k, k+1, \theta_k)\mathbf{b}$;// 6 flops
  $\mathbf{h}^k \leftarrow \mathbf{G}(k, k+1, \theta_k)\mathbf{h}^k$;// 6 flops, roll h_{k+1k} into the kk entry
  $\lambda^k \leftarrow$ Back Substitution$(\mathbf{R}^k, \mathbf{b})$;// $O(k^2)$ flops, at this point $\mathbf{R}^k = \left[\mathbf{R}^{k-1}, \mathbf{h}^k\right]$ is upper triangular
end for

2.3 Convergence behavior

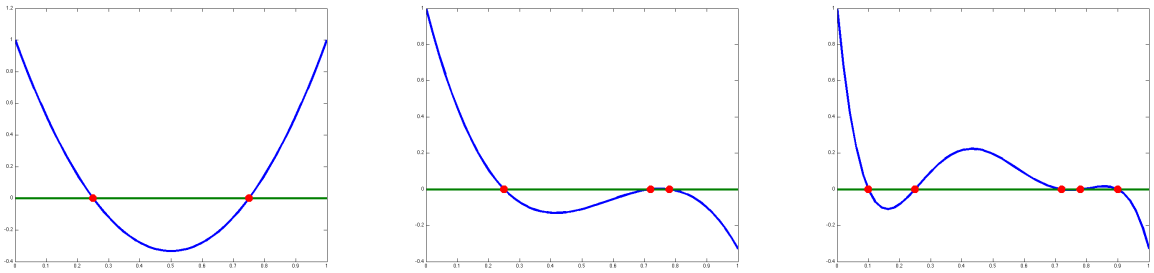

Figure 1: Minimal polynomials for the cases of 2,3 and 5 distinct eigenvalues. When the number of distinct eigenvalues is low, the Krylov method will converge very quickly because with each iteration our convergence will be gauged by the size of a polynomial over the spectrum of the matrix. In these cases, it is easy to construct a polynomial that vanishes over the spectrum with just a few iterations.

We showed previously that GMRES will converge as soon as we have taken a number of iterations equal to the number of distinct eigenvalues in the system (where I count an eigenvalue with a

non-trivial Jordan block (of size m) m times). In general, the convergence behavior will depend on the eigenstructure of the matrix. Specifically, we can look at the effect of the eigenstructure on the reduction of the residual at each iteration.

First, it is helpful to consider the polynomial nature of the Krylov method. The k^{th} iterate $\mathbf{x}^k$ is given by

$$\mathbf{x}^k = \underset{\mathbf{x} \in \mathcal{K}^k}{\text{argmin}} \ ||\mathbf{b} - \mathbf{A}\mathbf{x}||_2$$

where $\mathcal{K}^k = \text{span}\{\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A^2}\mathbf{b}, ..., \mathbf{A^{k-1}}\mathbf{b}\}$. In other words, $\mathbf{x} = \sum_{i=0}^{k-1} c_i \mathbf{A}^i \mathbf{b}$ so the L^2 norm of the residual is $||\mathbf{b} - \mathbf{A}\mathbf{x}||_2 = ||\mathbf{b} - \mathbf{A}\left(\sum_{i=0}^{k-1} c_i \mathbf{A}^i \mathbf{b}\right)||_2$. If we define the polynomial

$$p^k(\lambda) = 1 - \sum_{i=1}^{k} c_i \lambda^i$$

then the L^2 norm of the residual is $||\mathbf{b} - \mathbf{A}\mathbf{x}||_2 = ||p^k(\mathbf{A})\mathbf{b}||_2$. Therefore we can think of the k^{th} iterate as the solution to the problem of finding a polynomial p^k of degree k that minimizes the L^2 norm when evaluated at $\mathbf{A}$ and applied to $\mathbf{b}$ (also, we require that $p^k(0) = 1$). It can be shown that the minimizer p^k is the characteristic polynomial of the matrix $\hat{\mathbf{H}}^k = \left(\mathbf{Q}^k\right)^T \mathbf{A}\mathbf{Q}^k \in \mathbb{R}^{k \times k}$.

We can use this to understand how the residual ($\mathbf{r}^k = ||\mathbf{b} - \mathbf{A}\mathbf{x}^k||_2$) reduces in magnitude with each iteration. First, it should be clear from the formulation of the k^{th} iteration as a minimizer of the residual over a space that increases in dimension with iteration that the residual will decrease with k: $\mathbf{r}^{k+1} \leq \mathbf{r}^k$. However, we can also go further by considering polynomials that are small on $\mathbf{A}$. Specifically, we can say

$$\frac{||\mathbf{r}^k||_2}{||\mathbf{b}||_2} \leq \inf_{p^k \in \mathcal{P}^k} \ ||p^k(\mathbf{A})||_2$$

where $\mathcal{P}^k$ is the set of polynomials p of degree at most k with $p(0) = 1$. Now, if we let write $\mathbf{A} = \mathbf{V}\mathbf{J}\mathbf{V}^{-1}$ where $\mathbf{J}$ is the matrix of Jordan blocks then

$$\frac{||\mathbf{r}^k||_2}{||\mathbf{b}||_2} \leq ||\mathbf{V}||_2 ||\mathbf{V}^{-1}||_2 \inf_{p^k \in \mathcal{P}^k} \ ||p^k(\mathbf{J})||_2 = \kappa(\mathbf{V}) \inf_{p^k \in \mathcal{P}^k} \ ||p^k(\mathbf{J})||_2$$

Although in general, the super diagonal in the non-trivial Jordan blocks will complicate the argument a little, we can look at the diagonlizable case (i.e. no non-trivial Jordan blocks and $\mathbf{J} = \Lambda$). In this case, we can look at the minimization problem as the process of finding a polynomial that is small on the spectrum of $\mathbf{A}$. In other words, if we can find a polynomial that is small at every eigenvalue of $\mathbf{A}$ then the residual will decrease very rapidly with each iteration. The extreme case of this is when k is larger than the number of eigenvalues. In that case, the minimal polynomial will be zero at all of the eigenvalues and will be in $\mathcal{P}^k$. More commonly, we would hope to quickly (i.e. we would hope that they are in $\mathcal{P}^k$ with k small) find polynomials that are merely small at all the eigenvalues. This will be easy if the eigenvalues are close together, but more difficult when they are distinct and spaced far apart (see Figure 2). This condition with which the eigenvalues are clustered is clearly related to the condition number of $\mathbf{A}$.

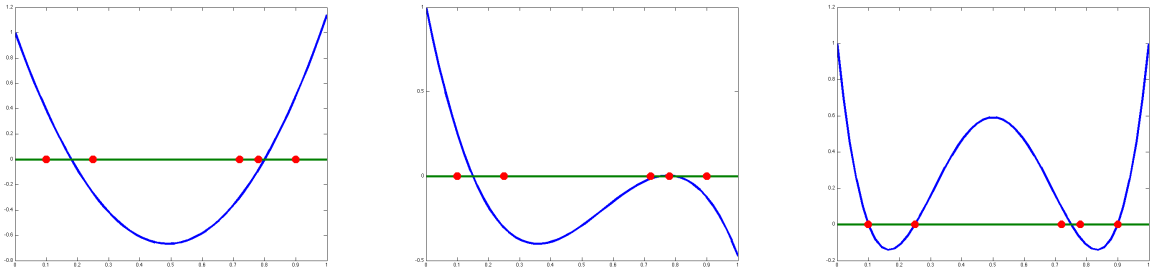

Figure 2: If we do not have enough flexibility in our choice of polynomial to construct one that vanishes at the eigenvalues, we can still construct one that is reasonably small if the eigenvalues are close enough together. The images show a few example polynomials that are relatively small on the spectrum, but that do not vanish. We would know that our residual would be reduced at this iteration by a factor no larger than the value of these polynomials over the spectrum. The polynomials are degree 2,3 and 4 (from left to right) and we would need 5 to vanish completely on the spectrum.