Numerical Methods for Polynomial Root-finding Problem

Jianxiang Fan and Nir Boneh

Polynomial Root-finding

$$p_n(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + x^n$$

- **1. Finding Eigenvalues of the Companion Matrix**
  - Francis's Algorithm with Double-shift
  - "Fast" QR Algorithm (*) (2010)
- **2. Iterative Approximation**
  - Newton-Horner Method
  - Muller's Method

Companion Matrix

$$p_n(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + x^n$$

$$A = \begin{bmatrix} 0 & & & & -a_0 \\ 1 & 0 & & & -a_1 \\ & 1 & \ddots & & \vdots \\ & & \ddots & 0 & -a_{n-2} \\ & & & 1 & -a_{n-1} \end{bmatrix}$$

Upper Hessenberg

The characteristic function of A is $p_n(x)$

Francis's Alg: Single-shift vs. Double-shift

- **Single shift**
  - Only practical when all eigenvalues are real
  - Complex shift μ => Complex matrix (A - μI)
- **Double shift**
  - Real or conjugate pairs
  - If ρ1 and ρ2 are conjugate pairs $(A - \rho_2 I)(A - \rho_1 I)$ is real

An Iteration of Double-shift Francis's

- **Step 1**
  - Pick shifts ρ1 and ρ2, which are eigenvalues of the 2×2 submatrix in the lower right corner of A
- **Step 2**
  - Don't need to compute $(A - \rho_2 I)(A - \rho_1 I)$
  - Just need the first column $x = p(A)e_1 = (A - \rho_2 I)(A - \rho_1 I)e_1.$ which only has nonzero entries in its first 3 positions.
  - Constant FLOPs

An Iteration of Double-shift Francis's (Cont.)

- **Step 3**
  - Compute a Householder reflector Q_0, such that $Q_0x = \alpha e_1$, where $\alpha = \pm\|x\|_2$
  - Since x only has nonzero entries in its first 3 positions, just need to compute a 3×3 householder reflector
  - Constant FLOPs

An Iteration of Double-shift Francis's (Cont.)

- **Step 4**
  - Use Q_0 to perform a similarity transformation: $A \Rightarrow Q_0^* A Q_0$
  - Combining 3 rows and then 3 columns, 15n + 27 FLOPs
  - Create the bulge

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ + & * & * & * & * & * \\ + & + & * & * & * & * \\ & & & * & * & * \\ & & & & * & * \end{bmatrix}$$

An Iteration of Double-shift Francis's (Cont.)

- **Step 5**
  - Bulge chasing $\hat{A} = Q_{n-2}^* \cdots Q_1^* Q_0^* A Q_0 Q_1 \cdots Q_{n-2}$

$$Q_k = \begin{bmatrix} I_k & & \\ & \tilde{Q}_k & \\ & & I_{n-k-3} \end{bmatrix} \qquad Q_{n-2} = \begin{bmatrix} I_{n-2} & \\ & \tilde{Q}_{n-2} \end{bmatrix}$$

$\tilde{Q}_k$ is 3×3 Householder reflector $\qquad \tilde{Q}_{n-2}$ is 2×2 Givens rotator

$$k = 0, 1, \cdots, n-3.$$

  - FLOPs: 15*n + 15*(n-1) + … + 15*4 = 15/2 * n^2

Deflation

- Doesn't always cause convergence to a triangular form
- Pairs of complex conjugate eigenvalues emerge in 2×2 blocks along the main diagonal of a block triangular matrix
- Check subdiagonal elements

$$|a_{k+1,k}| \leq u(|a_{kk}| + |a_{k+1,k+1}|)$$

“Fast” QR algorithms for Companion Matrices

- Double-shift Francis’s : O(n^2) FLOPs each iteration

- Hidden properties of companion matrix
- D. A. Bini, P. Boito, Y. Eidelman, L. Gemignani and I. Gohberg, ***A Fast Implicit QR Eigenvalue Algorithms for Companion Matrices***, Linear Algebra Appl., April (2010)
- O(n) FLOPs each iteration, with constant factor 243 theoretically

- Implement single-shift version based on their paper
- A brief introduction

H_n Class Matrix

$$A = \begin{bmatrix} 0 & & & & 1 \\ 1 & 0 & & & 0 \\ & 1 & \ddots & & \vdots \\ & & \ddots & 0 & 0 \\ & & & 1 & 0 \end{bmatrix} - \begin{bmatrix} p_0 + 1 \\ p_1 \\ \vdots \\ p_{n-1} \end{bmatrix} \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix}$$

$H \in \mathcal{H}_n$ if there exist $U \in \mathbb{C}^{n \times n}$ unitary and $\mathbf{z}, \mathbf{w} \in \mathbb{C}^n$ such that

$$H = U - \mathbf{z}\mathbf{w}^T$$

Generating Elements

Lemma 3.2: (Decomposition of U): If $A = U - \mathbf{z}\mathbf{w}^T \in \mathcal{H}_n$, there exist $n-2$ unitary matrix $\mathcal{V}_{n-1}, \mathcal{V}_{n-2}, \cdots, \mathcal{V}_2$, $n-2$ 3×3 unitary matrix $\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_{n-2}$ and β_j, such that

$$\beta_n = w_n, \begin{bmatrix} \beta_k \\ 0 \end{bmatrix} = \mathcal{V}_k^* \begin{bmatrix} w_k \\ \beta_{k+1} \end{bmatrix}, k = n-1, n-2, \cdots, 2$$

$$U = V_{n-1}V_{n-2}\cdots V_2 F_1 F_2 \cdots F_{n-2}$$

where

$$V_k = \begin{bmatrix} I_{k-1} & & \\ & \mathcal{V}_k & \\ & & I_{n-k-1} \end{bmatrix}, k = 2, \cdots, n-1$$

$$F_k = \begin{bmatrix} I_{k-1} & & \\ & \mathcal{F}_k & \\ & & I_{n-k-2} \end{bmatrix}, k = 1 \cdots, n-2$$

Generating Elements (Cont.)

$$
\begin{aligned}
&\beta_n = w_n \\
&\textbf{for } k = n-1 : -1 : 2 \\
&\qquad [\mathcal{V}_k, \beta_k] = \textbf{givens}(x_k, \beta_{k+1}) \\
&\qquad U(k : k+1, :) = \mathcal{V}_k^* U(k : k+1, :) \\
&\textbf{for } k = 1 : n-3 \\
&\qquad \mathcal{F}_k = \textbf{householder}(U(k : k+2, k)) \\
&\qquad U(k : k+2, k : n) = \mathcal{F}_k^* U(k : k+2, k : n) \\
&\mathcal{F}_{n-2} = U(n-2 : n, n-2 : n)
\end{aligned}
$$

Generating Elements are not easy to be manipulated under the QR iteration.

Upper generators

Theorem 3.3: Suppose $\{\mathcal{V}_k\}_2^{n-1}, \{\mathcal{F}_k\}_1^{n-2}, \mathbf{z}, \mathbf{w}$ are the generating elements of an $\mathcal{H}_n$ class matrix $A = U - \mathbf{z}\mathbf{w}^T \in \mathcal{H}_n$. The entries $u_{i,j}$, $\max\{1, i-2\} \leq j \leq n, 1 \leq i \leq n$, satisfy the following relations

$$u_{i,j} = \boldsymbol{g}_i^T B_{i,j}^{\times} \boldsymbol{h}_j \text{ for } j - i \geq 0$$

$$u_{i,j} = \sigma_j \text{ for } 1 \leq i = j+1 \leq n$$

where the vectors $\boldsymbol{h}_k$ and the matrices B_k are determined by the formulas

$$\boldsymbol{h}_k = \mathcal{F}_k(1:2, 1), \quad B_{k+1} = \mathcal{F}_k(1:2, 2:3), \quad 1 \leq k \leq n-2$$

and the vectors $\boldsymbol{g}_k$ and the numbers σ_k are computed recursively

$$\Gamma_1 = (0\,1), \quad \boldsymbol{g}_1^T = (1\,0)$$

$$\begin{bmatrix} \sigma_k & \boldsymbol{g}_{k+1}^T \\ * & \Gamma_{k+1} \end{bmatrix} = \mathcal{V}_{k+1} \begin{bmatrix} \Gamma_k & 0 \\ 0 & 1 \end{bmatrix} \mathcal{F}_k, \quad (k = 1, \cdots, n-2)$$

$$\sigma_{n-1} = \Gamma_{n-1} \boldsymbol{h}_{n-1}, \quad \boldsymbol{g}_n^T = \Gamma_{n-1} B_{n-1}$$

with the auxiliary variables $\Gamma_k \in \mathbb{C}^{1\times 2}$.

Iteration with Single-shift

Given the generating elements

$$\mathcal{V}_k \ (k = 2, \cdots, n-1),\ \mathcal{F}_k \ (k = 1, \cdots, n-2),\ \mathbf{z},\ \mathbf{w}$$

(1) Using algorithm from Theorem 3.3 compute upper generators $\boldsymbol{g}_i$, $\boldsymbol{h}_i$ $(i = 1, \cdots, n)$, B_k $(k = 2, \cdots, n)$ and σ_k $(k = 1, \cdots, n-1)$.

(2) Set $\beta_n = z_n$ and for $k = n-1, \cdots, 3$ compute

$$\beta_k = \mathcal{V}_k^*(1, 1:2) \begin{bmatrix} z_k \\ \beta_{k+1} \end{bmatrix}$$

(3) Using upper generators, σ_k and shift α to compute the Givens rotation matrices $\mathcal{G}_k$ $(k = 1, \cdots, n-1)$ and the updated perturbation vectors $\mathbf{z}^{(1)}$,$\mathbf{w}^{(1)}$

(4) Using β_k and the Givens rotation matrices $\mathcal{G}_k$ $(k = 1, \cdots, n-1)$ to compute the generating elements $\mathcal{V}_k^{(1)}$ $(k = 2, \cdots, n-1)$, $\mathcal{F}_k^{(1)}$ $(k = 1, \cdots, n-2)$

Try to implement in Matlab...

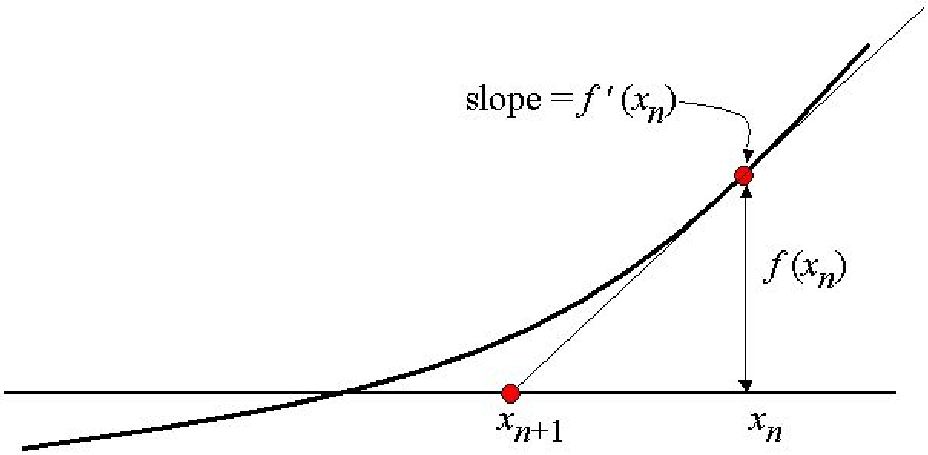
- **Single shift (OK)**
- **Deflation?**
    - Good news: Subdiagonal and diagonal elements can be represented by upper generators
    - Should reconstruct A to get sub-block? Become very unstable!
- **Double shift?**
- **Large constant factor, lots of memory manipulation...**

Newton-Horner Method

- An approximate polynomial root finding method
- Composed of two methods:
  - Newton Method
  - Horner Method

Newton's Method

- The Newton method takes a guess for the root of a function, x_0 and finds a better approximation with each iteration.



$$x_{n+1} = x_n - \frac{f'(x_n)}{f(x_n)}$$

Horner's Method

- Horner Method's objective is to find a solution to a polynomial given input x and minimize the amount of total flops.
- Horner states that polynomials can be rewritten as $p(x) = q(x) * (x - a) + c$ and if x equals a then the solution to the polynomial is c.
- For example $p(x) = x^3 - 2x^2 - 5x + 6$ can be rewritten as $p(x) = (x^2-5) * (x-2)-4$ and if $x = 2$ then, the solution is clearly -4.

Horner's Method Code

```
function [px, pprimex] = Horner(x)
    pprimex = 0.0
    px = c(1)
    for i = 2:n
      pprimex = pprimex * x + px
      px = px * x + c(i)
    end
```

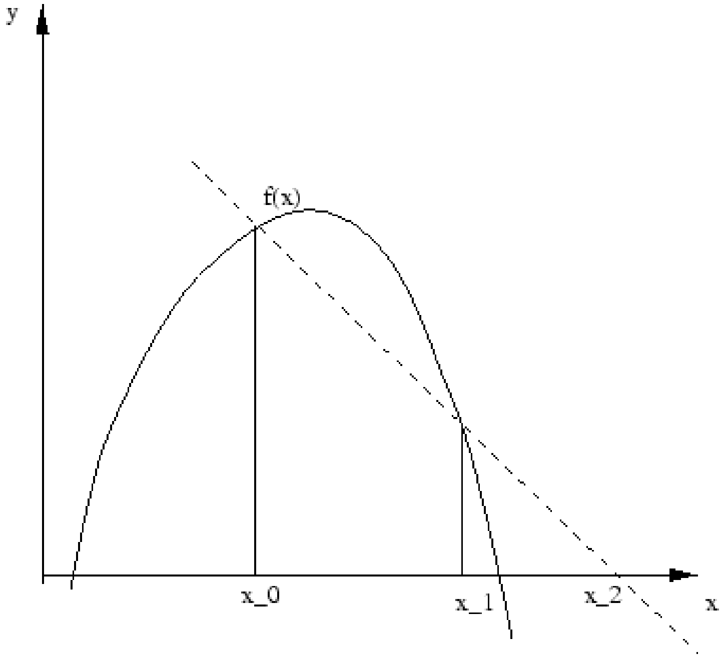
- It uses about 2n flops. Where a normal fully evaluated computation would use about n^2 /2 + n/2 flops

Deflation

- Combining both methods gives us a way to find one of the roots, but how do we find the others? Deflation!
- If you find root r1 for p(x) you can create a new polynomial p2(x) = p(x)/ (x - r1). p2(x) will contain all the original roots of p(x) except r1. Then you can continue with p3(x) and so on till pn(x) to find all roots.

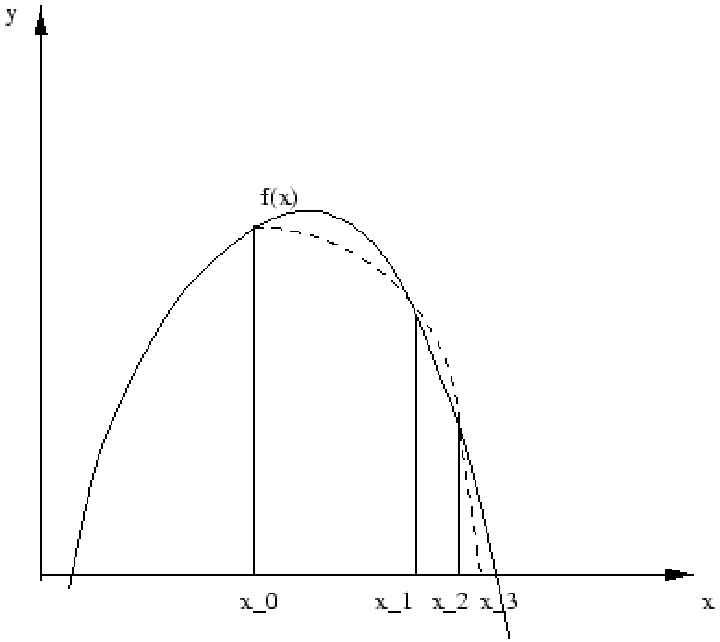
Muller's Method

- Based off the secant method, uses two initial guesses and linear interpolation.



Muller's Method

- Uses three initial guesses and quadratic interpolation.



Convergence Rate

- **Newton's Method**
  - Error in the kth iteration: $e_k = x_k - x^*$

$$e_{k+1} = \frac{f''(\xi_k)}{2f'(x_k)} e_k^2$$

Example f(x) = x^2 - 3

k	x_k	$\lvert e_k \rvert$
0	1.0	0.73205080756888
1	2.0	0.26794919243112
2	1.75	0.01794919243112
3	1.73214285714286	0.00009204957398
4	1.73205081001473	0.00000000244585

$$|f''(\sqrt{3})/2f'(\sqrt{3})| \approx 0.2886751$$

$$|e_4|/|e_3|^2 \approx 0.2886598$$

Convergence Rate (Cont.)

- ## Muller's Method
- Order of convergence: ~1.84

- ## Francis's Algorithm with double shift
- Shifts picking: eigenvalues of the 2×2 submatrix in the lower right corner of A
- Generally cubic, worst case quadratic

Conditioning of Polynomial Root-finding

- In general ill-conditioned
- Example : x^2 - 2x + 1 = 0 => x_1 = 1, x_2 = 1

  x^2 - 2.00001x + 1 = 0 => x_1 = 1.0032, x_2 = 0.9968

  Relative error in coefficient: 5 x 10^(-6)

  Relative error in roots: 0.0032
- Wilkinson's Polynomial: w(x) = (x-20)(x-19)...(x-1) = x^20 - 210x^19 + ...

  Perturb: -210 => -210 - 2^(-23)

  x_{16} = 16 => 16.73 + 2.81i
- Condition number of root x_i w.r.t the perturbation of a_i

$$\frac{|\delta x_j|}{|x_j|} \Big/ \frac{|\delta a_i|}{|a_i|} = \frac{|a_i x_j^{i-1}|}{|f'(x_j)|}$$

Cond_{16}: $O(10^{10})$

An Important Lesson

DON'T compute the eigenvalues of a matrix by finding coefficients of the characteristic polynomial, and then solving its roots by Newton-Horner.

Reference

[1] J. G. F. Francis, *The QR transformation. II*, Comput. J. 4 (1961/1962), 332-345.

[2] David S. Watkins, *Francis's Algorithm*, The American Mathematical Monthly 118(5), May (2011)

[3] D. A. Bini, Y. Eidelman, L. Gemignani and I. Gohberg, *Fast QR Eigenvalue Algorithms for Hessenberg Matrices Which Are Rank-One Perturbations of Unitary Matrices*, SIAM. J. Matrix Anal. & Appl., 29(2), (2007), 566-585.

[4] D. A. Bini, P. Boito, Y. Eidelman, L. Gemignani and I. Gohberg, *A Fast Implicit QR Eigenvalue Algorithms for Companion Matrices*, Linear Algebra Appl., April (2010)

[5] D. Bindel, S. Chandresekaran, J. Demmel, D. Garmire, and M. Gu, *A Fast and Stable Nonsymmetric Eigensolver for Certain Structured Matrices*, Technical report, University of California, Berkeley, CA, (2005)

[6] A. Quarteroni, R. Sacco, F. Saleri *Numerical Mathematics*, Second Edition, TAM37, Springer, Berlin (2007)

[7] Lloyd N. Trefethen, David Bau III, *Numerical Linear Algebra*, SIAM: Society for Industrial and Applied Mathematics (1997)

[8] Biswa Nath Datta, Numerical Linear Algebra and Applications, Second Edition, SIAM: Society for Industrial and Applied Mathematics (2010)