# Variable Precision Newton's Method to Solve Polynomial Systems

Jan Verschelde

University of Illinois at Chicago
Department of Mathematics, Statistics, and Computer Science
`http://www.math.uic.edu/~jan`
`jan@math.uic.edu`

Graduate Computational Algebraic Geometry Seminar

# Outline

1. Introduction
   - problem statement
2. Condition Numbers
   - linear systems
   - polynomial evaluation
3. Newton's Method in Variable Precision
   - relate precision to condition numbers
   - implementation in progress

# Variable Precision Newton's Method

1. Introduction
   - problem statement
2. Condition Numbers
   - linear systems
   - polynomial evaluation
3. Newton's Method in Variable Precision
   - relate precision to condition numbers
   - implementation in progress

# problem statement

Application of Newton's method:

Input: $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, a square polynomial system;
$\mathbf{z}_0$, an initial approximation for a root;
$d$, number of correct decimal places in the result.
Output: $\mathbf{z}$, $|\mathbf{z} - \mathbf{z}^*| \leq 10^{-d}$, where $\mathbf{f}(\mathbf{z}^*) = \mathbf{0}$.

Problem: decide the working precision to get the desired accuracy.

Let precision the precision be variable:

1. Double precision, $\epsilon_{\rm mach} = 2^{-53} \approx$ `1.110e-16`, in hardware.
2. Double double precision, $\epsilon_{\rm mach} = 2^{-104} \approx$ `4.930e-32`.
   Cost overhead is similar to the cost of complex arithmetic.
3. Quad double precision, $\epsilon_{\rm mach} = 2^{-209} \approx$ `1.215e-63`.
4. Arbitrary multiprecision is flexible, but has a high cost.

# references to the literature


- D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler: **Adaptive multiprecision path tracking.** *SIAM J. Numer. Anal.*, 46(2):722–746, 2008.
- J.W. Demmel: ***Applied Numerical Linear Algebra***. SIAM, 1997.
- G.H. Golub and C.F. Van Loan: ***Matrix Computations***. The Johns Hopkins University Press, third edition, 1996.
- N.J. Higham: ***Accuracy and Stability of Numerical Algorithms***. SIAM, 1996.

# numerical conditioning and variable precision

Condition numbers measure how sensitive

- the output of a numerical routine is,
- to changes in the input.

For example, assume

- the machine precision equals $10^{-16}$, and
- our problem has a condition number of $10^{8}$,

then the error on the output of a numerically stable algorithm to solve our problem can be as large as $10^{-8} = 10^{8} \times 10^{-16}$.

In general, the decimal logarithm of the condition number predicts the loss of the number of accurate decimal places.

Therefore, given a number of decimal places that should be correct, we estimate the condition number and then adjust the precision.

# singularities and variable precision

Consider
$$\left(x - \frac{1}{3}\right)^2 = x^2 - \frac{2}{3}x + \frac{1}{9}$$
$$= x^2 - 0.6666\ldots x + 0.1111\ldots$$
$$\approx x^2 - 0.6666x + 0.1111$$

Solving with `numpy.roots([1, -0.6666, 0.1111])` returns `array([ 0.3333+0.00333317j, 0.3333-0.00333317j])`.

Each time we recompute $\frac{2}{3}$ and $\frac{1}{9}$ in a higher precision, the numerical conditioning of the roots worsen. In the limit, the condition number becomes $\infty$.

For a badly scaled regular problem, the condition number is finite.

For a singular problem, estimates for the condition number grow as we increase the working precision, as the condition number is infinite.

# Variable Precision Newton's Method

1. Introduction
   - problem statement
2. Condition Numbers
   - linear systems
   - polynomial evaluation
3. Newton's Method in Variable Precision
   - relate precision to condition numbers
   - implementation in progress

# singular values

Let $A \in \mathbb{C}^{n \times n}$, the Singular Value Decomposition (SVD) of $A$ is

$$A = U \Sigma V^H, \quad U^H U = I, \quad V^H V = I, \quad \Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n),$$

where

- $U$ and $V$ are unitary (orthogonal) matrices, and
- the singular values of $A$ are sorted: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$.

If $\sigma_n > 0$, then $\sigma_n$ is the distance of $A$ to the closest singular matrix.

Distance is measured in the 2-norm: $||A||_2 = \max\limits_{||\mathbf{x}||_2 = 1} ||A\mathbf{x}||_2$.

The condition number of $A$ with respect to the 2-norm:

$$\mathrm{cond}_2(A) = ||A||_2 ||A^{-1}||_2 = \frac{\sigma_1}{\sigma_n}.$$

# estimating condition numbers

Computing the $\Sigma$ of a Golub-Reinsch SVD takes $4n^3$ operations.

LU decomposition (row reduction with pivoting) costs $\frac{2}{3}n^3$ operations.

Given $A \in \mathbb{C}^{n \times n}$, the LINPACK command `lufco` computes

1. an LU decomposition: $PA = LU$, $P$ is a permutation matrix,
2. then solve $U^H \mathbf{z} = \mathbf{d}$, $L^H \mathbf{y} = \mathbf{z}$, and $A\mathbf{x} = P^H \mathbf{y}$,

where the components $d_j$ of $\mathbf{d}$ are chosen in $\{-1, +1\}$ to make $||\mathbf{y}||_1$ large, at a cost of $4n^2$ operations.

Despite the existence of counterexamples, the estimator "*is regarded as being almost certain to produce an estimate correct to within a factor of 10 in practice*." [Higham, 1996].

Naturally, if the estimate exceeds $10^{+15}$, the outcome is no longer reliable when computing in double precision, ...
... the actual condition number could for example be $10^{+51}$.

# variable precision linear system solving

Input: $(A, \mathbf{b}) \in \mathbb{C}^{n \times n} \times \mathbb{C}^n$ defines a linear system $A\mathbf{x} = \mathbf{b}$,
$d$ is the number of decimal places wanted as correct.
Output: solution to $A\mathbf{x} = \mathbf{b}$, correct to $d$ decimal places.

Solving a linear system with variable precision:

1. Estimate the inverse $\kappa^{-1}$ of the condition number with `lufco`. Then $L = \log_{10}(\kappa^{-1})$ is the expected loss in accuracy.
   If $|L| \geq \log_{10}(|\epsilon_{\rm mach}|)$, then double the working precision and repeat the condition number estimation.
2. Set the working precision $\epsilon_{\rm mach}$ so that
   $$\log_{10}(|\epsilon_{\rm mach}|) + L \geq d.$$
3. Solve $A\mathbf{x} = \mathbf{b}$ in the right working precision.

# experimental setup

Let $L$ be the loss of decimal places:

$$\Sigma = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 10^{L/(n-1)} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 10^{(n-2)L/(n-1)} & 0 \\ 0 & 0 & \cdots & 0 & 10^{L} \end{bmatrix},$$

then $A = U\Sigma V^H$ for two random unitary matrices $U$ and $V$.

*The machine precision must be such that* $\log_{10}(|\epsilon_{\rm mach}|) > |L|$.

For $\mathbf{x} = (1, 1, \ldots, 1)$, compute $\mathbf{b} = A\mathbf{x}$.

As test $A\mathbf{x} = \mathbf{b}$, with $\mathrm{cond}_2(A) = 10^L$ and known solution.

# polynomial evaluation

Let $f \in \mathbb{C}[\mathbf{x}]$, a polynomial in $n$ variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$:

$$f(\mathbf{x}) = \sum_{\mathbf{a} \in A} c_{\mathbf{a}} \mathbf{x}^{\mathbf{a}}, \quad c_{\mathbf{a}} \in \mathbb{C} \setminus \{0\}, \quad \mathbf{x}^{\mathbf{a}} = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}.$$

Measuring the sensitivity of evaluating the polynomial $f$ at $\mathbf{z} \in \mathbb{C}^n$:

$$\text{the condition number is } \mathrm{cond}(f, \mathbf{z}) = \frac{\displaystyle\sum_{\mathbf{a} \in A} |c_{\mathbf{a}}||\mathbf{z}^{\mathbf{a}}|}{|f(\mathbf{z})|}.$$

Factors that determine the magnitude of $\mathrm{cond}(f, \mathbf{z})$:

1. the magnitude of the coefficients $|c_{\mathbf{a}}|$,
2. the magnitude of the coordinates of $\mathbf{z}$: $|z_i|$, $i = 1, 2, \ldots, n$,
3. the largest degree of the monomials $a_1 + a_2 + \cdots + a_n$,
4. the distance of $\mathbf{z}$ to a root, $f(\mathbf{z}) \approx 0$.

# experimental setup

Making a polynomial $f$ with prescribed condition number, for evaluating $f$ at $\mathbf{z}$, choose the following factors:

1. $M_{\rm cf}$ is the magnitude of coefficients of $f$: $M_{\rm cf} \geq |c_{\mathbf{a}}|$,
2. $M_{\rm co}$ is the magnitude of the coordinates of $\mathbf{z}$: $M_{\rm co} \geq |z_i|$,
3. $d$ is the degree of the polynomial $f$,
4. $\delta$ is the distance of $\mathbf{z}$ to a root, change $f(\mathbf{x})$ into $f(\mathbf{x}) - f(\mathbf{z}) + \delta$.

Then the condition number can be as large as

$$\frac{M_{\rm cf} \times M_{\rm co}^d}{\delta}.$$

# an expression motivating interval arithmetic

Problem: Evaluate $f(x,y) =$

$$(333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

at $(77617, 33096)$.

An example of Stefano Taschini: ***Interval Arithmetic: Python Implementation and Applications.*** In the Proceedings of the 7th Python in Science Conference (SciPy 2008).

Siegfried M. Rump: **Verification methods: Rigorous results using floating-point arithmetic.** *Acta Numerica* 19:287-449, 2010.

***Problem: when does the precision become sufficient?***

# condition numbers at variable precision

The expresssion in the string

```
(333.75 - x**2)*y**6 + x**2*(11*x**2*y**2 - 121*y**4 - 2)
+ 5.5*y**8 + (1/2)*x*y^-1;
```

is parsed in to a Laurent polynomial (double precision format):

```
- x^2*y^6 + 5.50000000000000E+00*y^8 + 11*x^4*y^2 - 121*x^2*y^4
+ 3.33750000000000E+02*y^6 - 2*x^2 + 5.00000000000000E-01*x*y^-1
```

`rco` = inverse of condition number

| precision | rco | value |
|---|---|---|
| double precision | 6.494E-17 | -1.02823048247338E+21 |
| double double precision | 5.225E-38 | -8.27396059946821E-01 |
| quad double precision | 5.225E-38 | -8.27396059946821E-01 |
| 24 decimal places | 3.452E-25 | 5.46645820262317E+12 |
| 30 decimal places | 1.501E-32 | 2.37695172603940E+05 |
| 40 decimal places | 5.225E-38 | -8.27396059946821E-01 |

# Variable Precision Newton's Method

1. Introduction
   - problem statement

2. Condition Numbers
   - linear systems
   - polynomial evaluation

3. Newton's Method in Variable Precision
   - relate precision to condition numbers
   - implementation in progress

# Newton's method in variable precision

Denote by $J_{\mathbf{f}}(\mathbf{x})$ the Jacobian matrix of the system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ at $\mathbf{x}$.

Apply Newton's method on $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, at $\mathbf{z}_k$:

$$J_{\mathbf{f}}(\mathbf{z}_k)\Delta\mathbf{z} = -\mathbf{f}(\mathbf{z}_k), \quad \mathbf{z}_{k+1} := \mathbf{z}_k + \Delta\mathbf{z}.$$

Estimate condition numbers:

1. $L_1 = \log_{10}(\mathrm{cond}(J_{\mathbf{f}}(\mathbf{z}_k))$ loss when solving linear system;
2. $L_2 = \log_{10}(\mathrm{cond}(\mathbf{f}, \mathbf{z}_k))$, loss when evaluating system, where $\mathrm{cond}(\mathbf{f}, \mathbf{z}_k) = \max_{i=1}^{n} \mathrm{cond}(f_i, \mathbf{z}_k)$.

Then $L = \max(L_1, L_2)$ is the estimated loss of decimal places.

# experimental setup

For testing, we want a Jacobian matrix with given condition. Making a polynomial $f$ with prescribed gradient. Consider:

$$f(\mathbf{x}) = g(\mathbf{x}) + \sum_{k=1}^{n} c_k x_k + c_0,$$

where $g$ contains no linear or constant terms.

Let $v_\ell$ be the $\ell$-th value of the gradient of $f$: $v_\ell = \dfrac{\partial f}{\partial x_\ell}(\mathbf{z})$.

$$v_\ell = \frac{\partial f}{\partial x_\ell}(\mathbf{z}) = \frac{\partial g}{\partial x_\ell}(\mathbf{z}) + c_\ell \quad \Rightarrow \quad c_\ell = v_\ell - \frac{\partial g}{\partial x_\ell}(\mathbf{z})$$

Then $v_0 = f(\mathbf{z}) = g(\mathbf{z}) + \sum_{k=1}^{n} c_k z_k + c_0 \Rightarrow c_0 = v_0 - g(\mathbf{z}) - \sum_{k=1}^{n} c_k z_k$.

# implementation in progress

Current `newton_step` in `phcpy.solver`:

```
sols = newton_step(p,sols,precision='d')
                          precision='dd')
                          precision='qd')
                          precision='mp'decimals=100)
```

The goal is to provide a prototype like

```
sols = newton_step(p,sols,accuracy=8)
```