ROOT-FINDING METHODS FOR SOLVING DISPERSION EQUATIONS

Petr Hora

Institute of Thermomechanics AS CR, v. v. i.
Prague, Czech Republic

ENGINEERING MECHANICS 2016, Mini-symposium WAVES

May 9 – 12, 2016 Svratka Czech Republic

Motivation: Bar impact problem

To calculate the stress wave propagation in a bar impact problem it is used the integration along the dispersion curves.

This dispersion relations $f(x, \gamma a)$ is defined as

$$\left(2 - x^2\right)^2 J_0(\gamma a A)\, J_1(\gamma a B) + 4AB J_1(\gamma a A)\, J_0(\gamma a B) - \frac{2x^2}{\gamma a} A J_1(\gamma a A)\, J_1(\gamma a B) = 0,$$

where

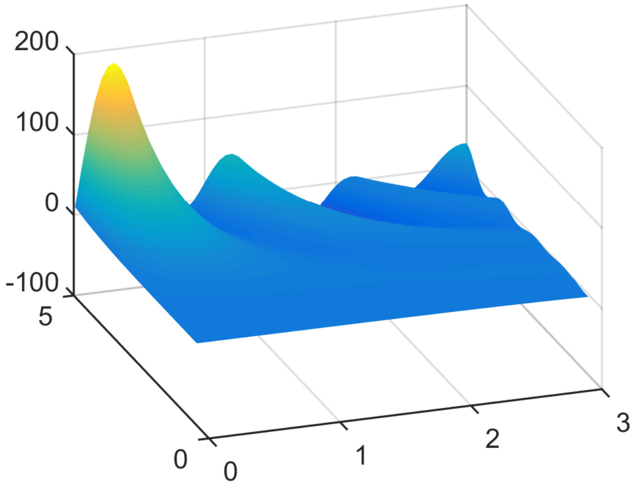
a	radius of the semi-infinite bar,
γ	wavenumber,
x	the ratio of the phase velocity and the shear wave velocity,
κ	the ratio of the squares of the phase velocities,
A	$\sqrt{\kappa x^2 - 1}$,
B	$\sqrt{x^2 - 1}$,
J	the Bessel function of the first kind.

Transcendental equations

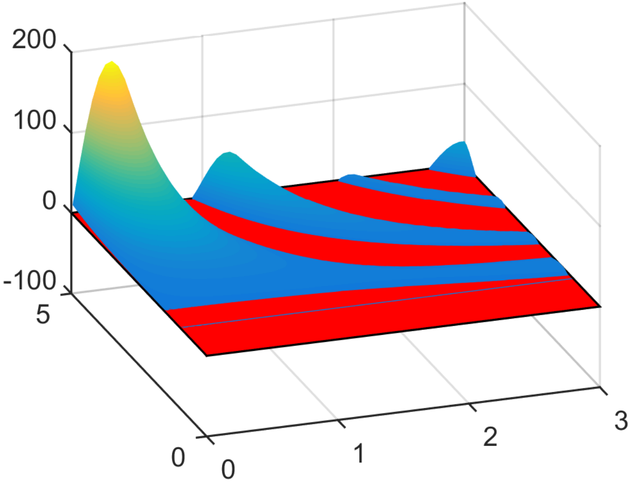
- A transcendental equation is an equation containing a transcendental function of the variable(s) being solved for.
- The most familiar transcendental functions are:
  - the logarithm,
  - the exponential (with any non-trivial base),
  - the trigonometric,
  - the hyperbolic functions,
  - and the inverses of all of these.
- Less familiar are:
  - The special functions of analysis, such as the gamma, elliptic, and zeta functions, all of which are transcendental.
  - The generalized hypergeometric and Bessel functions are transcendental in general, but algebraic for some special parameter values.

WIKIPEDIA, Transcendental function

$f(x, \gamma a)$



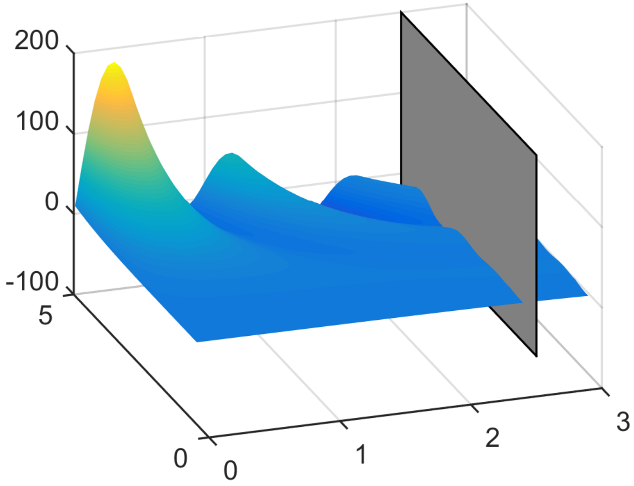
Flooded $f(x, \gamma a)$



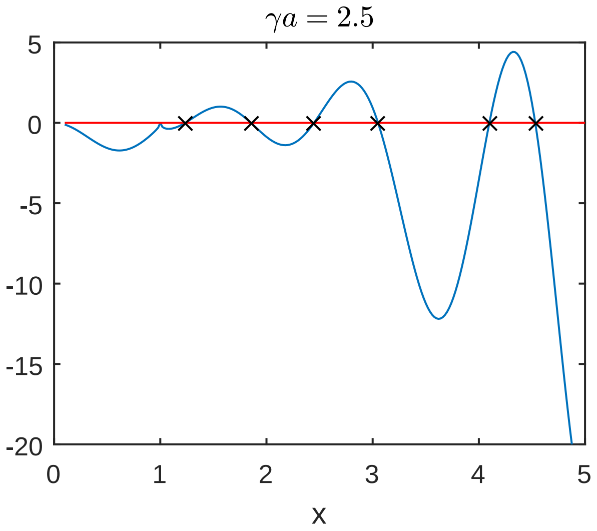
Solution methods

1. Root-finding
2. Chebyshev interpolation
3. Interval arithmetics
4. Marching squares
5. ...

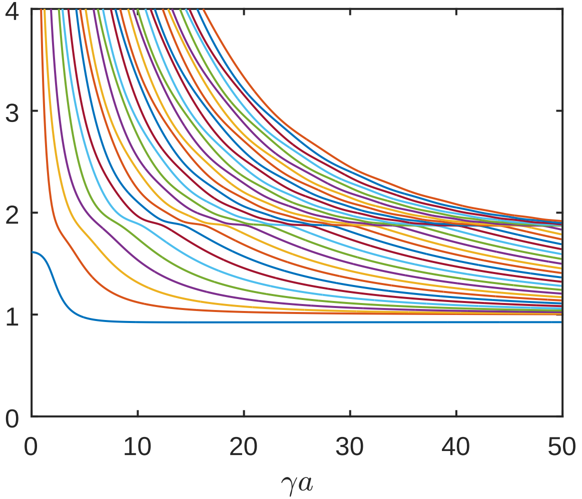
Root-finding, 3D



Root-finding, cut



Root-finding, DC



Root-finding, limits

1. $0 < x < 1$

$$\lim_{\gamma a \to +\infty} x = 0.92741271$$

2. $1 < x < 1/\sqrt{\kappa}$

$$\lim_{\gamma a \to 0_+} x = 1.61245155$$

3. $x > 1/\sqrt{\kappa}$

$$\lim_{\gamma a \to 0_+} x = \begin{cases} J_1(\gamma a\, x) = 0, \\ \left[\gamma a\, x\, J_0\left(\sqrt{\kappa}\, \gamma a\, x\right) - 2\sqrt{\kappa} J_1\left(\sqrt{\kappa}\, \gamma a\, x\right)\right] = 0. \end{cases}$$

Root-finding, Algorithm

- Stepping in γa,
- The guess for the first steps by means of the limit,
- The guess for the next steps by means of the extrapolation,
- Newton method,
- Parallel processing.

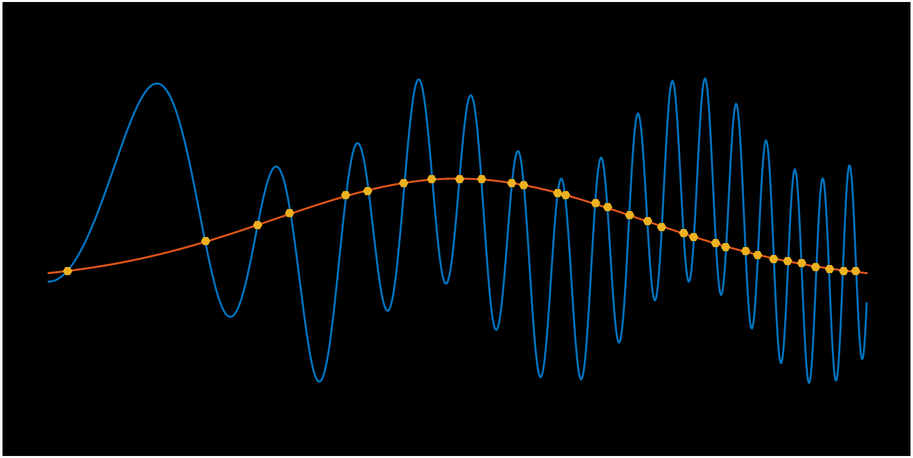
Root-finding, Algorithm

- Stepping in γa,
- The guess for the first steps by means of the limit,
- The guess for the next steps by means of the extrapolation,
- Newton method,
- Parallel processing.

Disadvantages

- Root skip,
- Lazy (particularly for random γa),
- Need of differentiation (Newton method).

Chebyshev interpolation



Chebyshev interpolation

```
% Define two functions
f = chebfun(@(x) sin(x.^2)+sin(x).^2, [0,10]);
g = chebfun(@(x) exp(-(x-5).^2/10), [0,10]);

% Compute their intersections
rr = roots(f - g);

% Plot the functions
plot([f g]), hold on

% Plot the intersections
plot(rr, f(rr), 'o')
```

Chebyshev interpolation

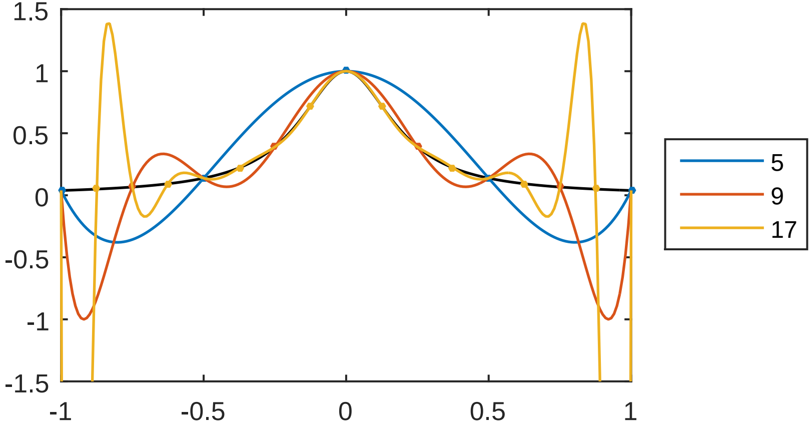
Solve-the-proxy methods in one unknown:

Approximation	Name
Piecewise linear interpolation	Make-a-Graph-Stupid algorithm
Linear Taylor series	Newton–Raphson iteration
Linear interpolant	secant iteration
Quadratic Taylor series	Cauchy's method
Quadratic interpolant	Muller's method
Inverse quadratic interpolant	Brent's algorithm
(Linear-over-linear) Padé approximant	Halley's scheme
(Quadratic-over-quadratic) Padé approximant	Shafer's method
Chebyshev polynomial interpolant	**Chebyshev proxy method**

J.P. Boyd: Finding the Zeros of a Univariate Equation, SIAM Rev., 55(2)

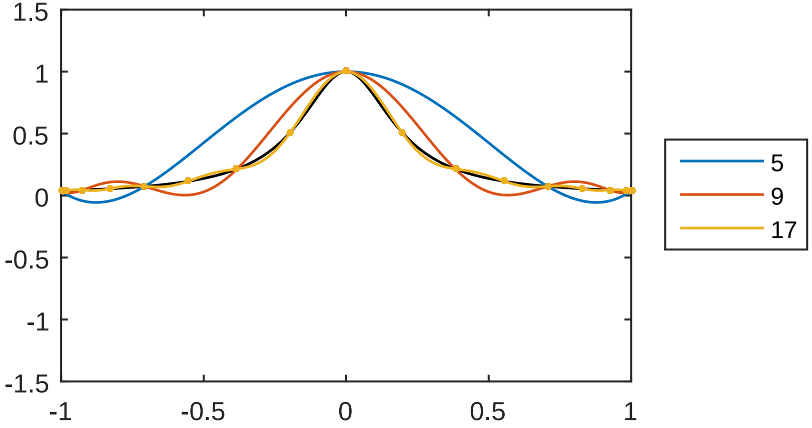
The Runge's phenomenon

- Problem of oscillation at the edges of an interval that occurs when using polynomial interpolation with polynomials of high degree over a set of equispaced interpolation points.



The Runge's phenomenon

- Problem of oscillation at the edges of an interval that occurs when using polynomial interpolation with polynomials of high degree over a set of equispaced interpolation points.



Chebyshev polynomials of the first kind, I

- The recurrence relation

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x). \end{aligned}$$

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 4x^3 - 3x \\ T_4(x) &= 8x^4 - 8x^2 + 1 \\ T_5(x) &= 16x^5 - 20x^3 + 5x \\ T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1 \end{aligned}$$

Chebyshev polynomials of the first kind, II

- Trigonometric definition

$$T_n(x) = \cos(n \arccos x) = \cosh(n \operatorname{arcosh} x)$$

- Roots
  n different simple roots, called Chebyshev roots, in the interval $[-1, 1]$.

$$x_k = \cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \dots, n$$

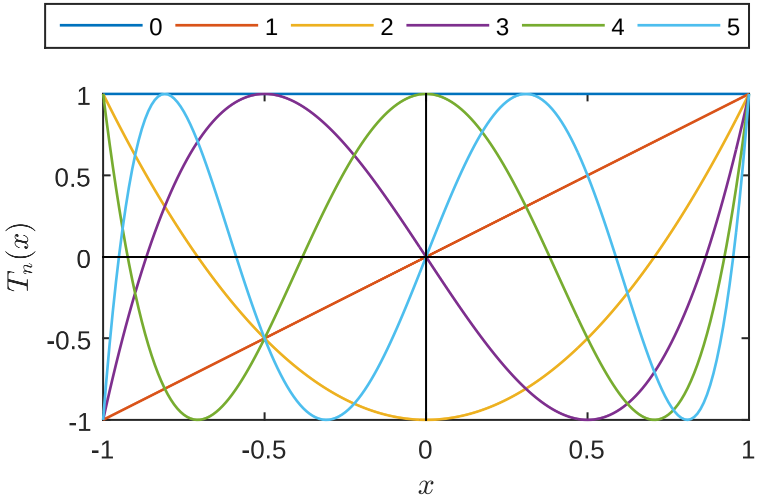
- Extrema

$$x_k = \cos\left(\tfrac{k}{n}\pi\right), \quad k = 0, \dots, n$$

  All of the extrema have values that are either -1 or 1.
  Extrema at the endpoints, given by:

$$\begin{aligned} T_n(1) &= 1 \\ T_n(-1) &= (-1)^n \end{aligned}$$

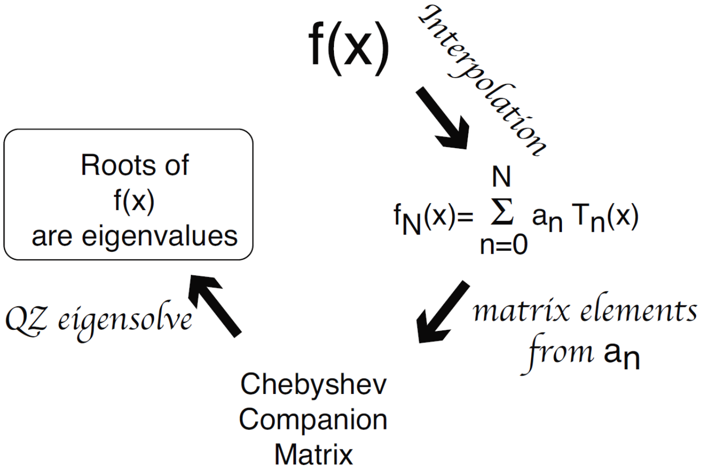
Chebyshev polynomials of the first kind, III



Chebyshev interpolation

- Basic idea:
  Represent functions using interpolants through (suitably rescaled) Chebyshev nodes
  $$x_j = -\cos\left(\frac{j\pi}{n}\right), \qquad 0 \leqslant j \leqslant n.$$
- Such interpolants have excellent approximation properties.
- Interpolants are constructed adaptively, more and more points used, until coefficients in Chebyshev series fall below machine precision.

Chebyshev-proxy rootfinder


$f(x)$
Interpolation
$f_N(x) = \sum_{n=0}^{N} a_n T_n(x)$
matrix elements from a_n
Chebyshev Companion Matrix
QZ eigensolve
Roots of f(x) are eigenvalues

Chebyshev interpolation, Algorithm I

1. Choose the following:
   1.1 γa
   1.2 Search interval, $x \in [a, b]$.
   The search interval must be chosen by physical and mathematical analysis of the individual problem. The choice of the search interval $[a, b]$ depends on the user's knowledge of the physics of his/her problem, and no general rules are possible.
   1.3 The number of grid points, N.
   N may be chosen by setting $N = 1 + 2^m$ and the increasing N until the Chebyshev series displays satisfactory convergence. To determine when N is sufficiently high, we can examine the Chebyshev coefficients a_j, which decrease exponentially fast with j.
2. Compute a Chebyshev series, including terms up to and including T_N, on the interval $x \in [a, b]$.
   2.1 Create the interpolation points (Lobatto grid):

$$x_k \equiv \frac{b-a}{2} \cos\left(\pi \frac{k}{N}\right) + \frac{b+a}{2}, \quad k = 0, 1, 2, \ldots, N.$$

Chebyshev interpolation, Algorithm II

2.2 Compute the elements of the $(N+1) \times (N+1)$ interpolation matrix. Define $p_j = 2$ if $j = 0$ or $j = N$ and $p_j = 1, j \in [1, N-1]$. Then the elements of the interpolation matrix are

$$I_{jk} = \frac{2}{p_j p_k N} \cos\left(j\pi \frac{k}{N}\right).$$

2.3 Compute the grid-point values of $f(x)$, the function to be approximated:

$$f_k \equiv f(x_k), \quad k = 0, 1, \dots, N.$$

2.4 Compute the coefficients through a vector-matrix multiply:

$$a_j = \sum_{k=0}^{N} I_{jk} f_k, \quad j = 0, 1, 2, \dots..., N.$$

The approximation is

$$f_N \approx \sum_{j=0}^{N} a_j T_j \left(\frac{2x - (b+a)}{b-a}\right) = \sum_{j=0}^{N} a_j \cos\left\{j \arccos\left(\frac{2x - (b+a)}{b-a}\right)\right\}.$$

Chebyshev interpolation, Algorithm III

3. Compute the roots of f_N as eigenvalues of the Chebyshev–Frobenius matrix.
   Frobenius showed that the roots of a polynomial in monomial form are also the eigenvalues of the matrix which is now called the *Frobenius companion matrix*. Day and Romero developed a general formalism for deriving the *Frobenius matrix* for any set of orthogonal polynomials.
4. Refine the roots by a Newton iteration with $f(x)$ itself.
   Once a good approximation to a root is known, it is common to *polish* the root to close to machine precision by one or two Newton iterations.

Chebyshev interpolation, Numerical experiments

- ApproxFun (Julia package)

  URL: github.com/ApproxFun/ApproxFun.jl.git

- CHEBFUN (MATLAB toolbox)

  Driscoll, T.A., Hale, N., Trefethen, L.N., editors, Chebfun Guide, Pafnuty Publications, Oxford, 2014.

Arbitrary precision computations

Evaluate

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

at $(77617, 33096)$.

single precision	$f \approx 1.172603\ldots$
double precision	$f \approx 1.1726039400531\ldots$
extended precision	$f \approx 1.172603940053178\ldots$
the true value	$f = -0.827386\ldots$

📕 Taschini, S.: Interval Arithmetic: Python Implementation and Applications. In the Proceedings of the 7th Python in Science Conference (SciPy 2008).

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

Multiprecision interval arithmetic

```
from sympy.mpmath import iv
print 'using 35 decimal places ...'
iv.dps = 35
iv_f = lambda x,y: (iv.mpf('333.75') \
- x**2)*y**6 \
+ x**2*(iv.mpf('11')*x**2*y**2 \
- iv.mpf('121')*y**4 - iv.mpf('2')) \
+ iv.mpf('5.5')*y**8 + x/(iv.mpf('2')*y);
iv_a = iv.mpf(str(a))
iv_b = iv.mpf(str(b))
iv_z = iv_f(iv_a,iv_b)
print iv_z
```

shows

```
using 35 decimal places ...
[-6.827396059946821368141165095479816292382, \
1.172603940053178631858834904520183709123]
```

```
print 'using 36 decimal places ...'
iv.dps = 36
...
```

shows

```
using 36 decimal places ...
[-0.827396059946821368141165095479816292005549, \
-0.827396059946821368141165095479816291817 41]
```

$$= -(0.827396059946821368141165095479816\overset{200549}{\underset{181741}{29}})$$

width of interval = upper bound on error

Interval Analysis

Computing with intervals began:
Bounding rounding and truncation errors in finite precision arithmetic
Ramon Moore, 1966, Interval Analysis

One of Methods of Representing Uncertainty.

Uncertain parameters are described by an upper and lower bound,
then rigorous bounds on the response are computed using
interval arithmetic
and
interval functions.

Moore, R. E.: Interval analysis.
Prentice-Hall series in automatic computation.
Englewood Cliffs, N.J.: Prentice-Hall, 1966.

Interval Arithmetic

Upper and Lower bound

$$[a, b] = x | a \leqslant x \leqslant b$$

Midpoint and Radius

$$[x_0, r] = x | x_0 - r \leqslant x \leqslant x_0 + r$$

Degenerate Interval - Scalar

$$x = [a, a]$$

Width

$$w([a, b]) = b - a$$

Magnitude

$$|[a, b]| = \mathsf{max}(|a|, |b|)$$

Midpoint

$$\mathsf{mid}([a, b]) = (a + b)/2$$

Equality

$$[a, b] = [c, d] \Longleftrightarrow a = b, c = d$$

Ordering

$$[a, b] < [c, d] \Longleftrightarrow b < c$$

Interval Arithmetic

Addition

$$[a, b] + [c, d] = [a + c, b + d]$$

Subtraction

$$[a, b] - [c, d] = [a - d, b - c]$$

Multiplication

$$[a, b] \times [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$$

Division

$$[a, b] \div [c, d] = [a, b] \times \left[\frac{1}{d}, \frac{1}{c}\right]$$

provided $[c, d]$ does not contain 0

Theorem (Ratz, 1996)

Let $\langle a, b\rangle$ and $\langle c, d\rangle$ be two non-empty bounded real intervals.
Then

$$\langle a, b\rangle \div \langle c, d\rangle = \begin{cases} \langle a, b\rangle \times \langle 1/d, 1/c\rangle & \text{if } 0 \notin \langle c, d\rangle \\ \langle -\infty, \infty\rangle & \text{if } 0 \in \langle a, b\rangle \wedge 0 \in \langle c, d\rangle \\ \langle b/c, \infty\rangle & \text{if } b < 0 \wedge c < d = 0 \\ \langle -\infty, b/d\rangle \cup \langle b/c, \infty\rangle & \text{if } b < 0 \wedge c < 0 < d \\ \langle -\infty, b/d\rangle & \text{if } b < 0 \wedge 0 = c < d \\ \langle -\infty, a/c\rangle & \text{if } 0 < a \wedge c < d = 0 \\ \langle -\infty, a/c\rangle \cup \langle a/d, \infty\rangle & \text{if } 0 < a \wedge c < 0 < d \\ \langle a/d, \infty\rangle & \text{if } 0 < a \wedge 0 = c < d \\ \emptyset & \text{if } 0 \notin \langle a, b\rangle \wedge c = d = 0 \end{cases}$$

Ratz, D.: On extended interval arithmetic and inclusion isotonicity. Institut für Angewandte Mathematik, Universität Karlsruhe, 1996.

Anomalies in Interval Arithmetic

Subdistributivity

$$[a, b] \times ([c, d] \pm [e, f]) \subseteq [a, b] \times [c, d] \pm [a, b] \times [e, f]$$

Subcancelation

$$[a, b] - [c, d] \subseteq ([a, b] + [e, f]) - ([c, d] + [e, f])$$

$$[a, b] \div [c, d] \subseteq ([a, b] \times [e, f]) \div ([c, d] \times [e, f])$$

$$0 \in [a, b] - [a, b]$$

$$1 \in [a, b] \div [a, b]$$

Images of Functions

Monotone functions If $f(x) : x \to \mathbb{R}$ is non-decreasing, then

$$f([a, b]) = [f(a), f(b)].$$

Examples:

$$exp([a, b]) = [exp(a), exp(b)]$$

$$\log([a, b]) = [log(a), log(b)]$$

Some basic functions Images $x^2, sin(x), \ldots$, are easily calculated, too.

Example:

$$[a, b]^2 = \begin{cases} \left[\min(a^2, b^2), \max(a^2, b^2)\right], & \text{if } 0 \notin [a, b], \\ \left[0, \max(a^2, b^2)\right], & \text{otherwise.} \end{cases}$$

More complex functions Bessel, ...

Goal of Interval Computation

To compute the SHARPEST possible interval solution set which COMPLETELY CONTAINS the true solution set.

Rigor is easy to accomplish - $[-\infty, \infty]$ always true but useless

Tight solutions are difficult - primarily due to dependence

The Dependence Problem

Interval arithmetic implicitly assumes
each occurrence of a variable is independent.

Example: Compute X^2 without the power rules where $X = [-1, 2]$.

$$X^2 = X \times X = [\min(-2, 1, 4), \max(-2, 1, 4)] = [-2, 4]$$

The answer should be $X^2 = [0, 4]$

$$\Downarrow$$

The excess width is know as the dependence problem.

If an interval variable appears only once in an expression
no widening of the interval occurs.

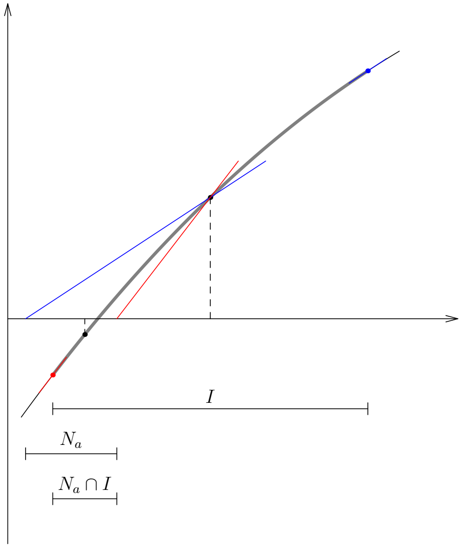
Solving the Dependence Problem

1. Analytically rewrite expressions to minimize the number of times a variable occurs.
2. Delay interval computations as late as possible
3. Reduce dependencies computationally (e.g. with automatic differentiation)

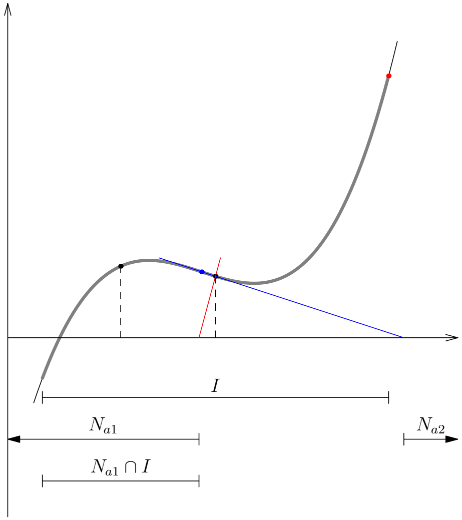
Programming with Intervals

- Matlab
  - INTLAB - pure Matlab
    - interval arithmetic with real and complex data including scalars, vectors, matrices and sparse matrices
    - automatic differentiation
    - rigorous interval standard functions
    - rigorous input and output
    - multiple precision interval arithmetic
  - b4m - uses C interval library, BIAS
- C/C++ and Fortran 77/90 - BIAS, PROFIL, C-XSC, INTLIB, GlobSol
- Python - interval, mpmath(iv)
- Julia - ValidatedNumerics.jl, Intervals.jl
- Maple - intpak, intpakX
- Mathematica

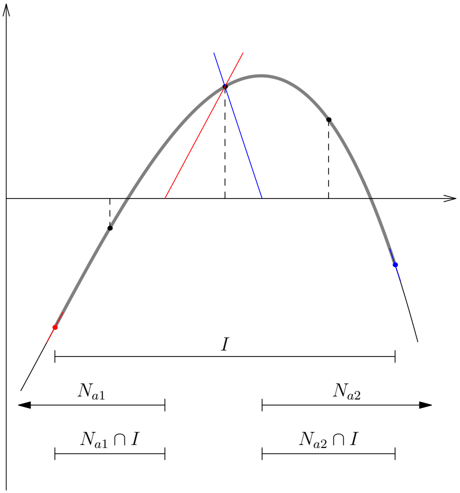
Interval Newton Method



Interval Newton Method



Interval Newton Method



Automatic Differentiation (AD)

- is a set of techniques based on the mechanical application of the chain rule to obtain derivatives of a function given as a computer program.

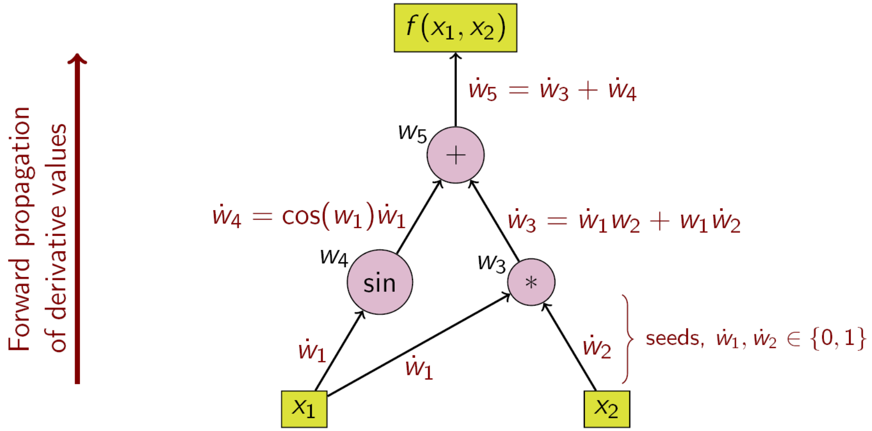
AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations such as additions or elementary functions such as exp(). By applying the chain rule of derivative calculus repeatedly to these operations, derivatives of arbitrary order can be computed automatically, and accurate to working precision.

Conceptually, AD is different from symbolic differentiation and approximations by divided differences.

AD is used in the following areas:

- Numerical Methods
- Sensitivity Analysis
- Design Optimization
- Data Assimilation & Inverse Problems

$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$



$f(x_1, x_2)$
$\dot{w}_5 = \dot{w}_3 + \dot{w}_4$
w_5
$+$
$\dot{w}_4 = \cos(w_1)\dot{w}_1$
$\dot{w}_3 = \dot{w}_1 w_2 + w_1 \dot{w}_2$
w_4
sin
w_3
$*$
seeds, $\dot{w}_1, \dot{w}_2 \in \{0, 1\}$
$\dot{w}_1$
$\dot{w}_1$
$\dot{w}_2$
x_1
x_2
Forward propagation of derivative values

By Berland at en.wikipedia [Public domain], from Wikimedia Commons

```
function   hodnota     DispRov3 GammaD Zeta  Poisson

hodnota NaN

while

    k       *Poisson     *    Poisson

    GammaD scal2matr GammaD Zeta
    Zeta scal2matr Zeta GammaD

    if  size GammaD    size Zeta
        hodnota hodnota *ones size GammaD
    else
        disp 'GammaD a Zeta musí mít stejnou velikost'
        break
    end

    if  all all GammaD          all all Zeta
        Zeta    Zeta      sqrt k      NaN  %Hodnoty přesahující vymezený obor hodnot jsou ignorovány.

        hodnota   Zeta          *sin GammaD *sqrt Zeta         *cos GammaD *sqrt k *Zeta
           *sqrt Zeta        *sqrt k *Zeta        *cos GammaD *sqrt Zeta         *sin GammaD *sqrt k *Zeta

    else
        disp 'GammaD a Zeta musí být matice kladných reálných čísel (intervalů)'
        break
    end

    break
end
end
```

Marching Squares

- is a computer graphics algorithm that generates contours for a two-dimensional scalar field (rectangular array of individual numerical values).

Typical applications include the Contour lines on topographic maps
or the generation of isobars for weather maps.

The algorithm is embarrassingly parallel,
because all cells are processed independently.

WIKIPEDIA, Marching Squares

Thank you for your attention!

Any questions?

The work was supported by the institutional support RVO: 61388998.

Contents