Efficient approximation of polynomials

Guillaume Moroz

September 25, 2023

IHP

Problems

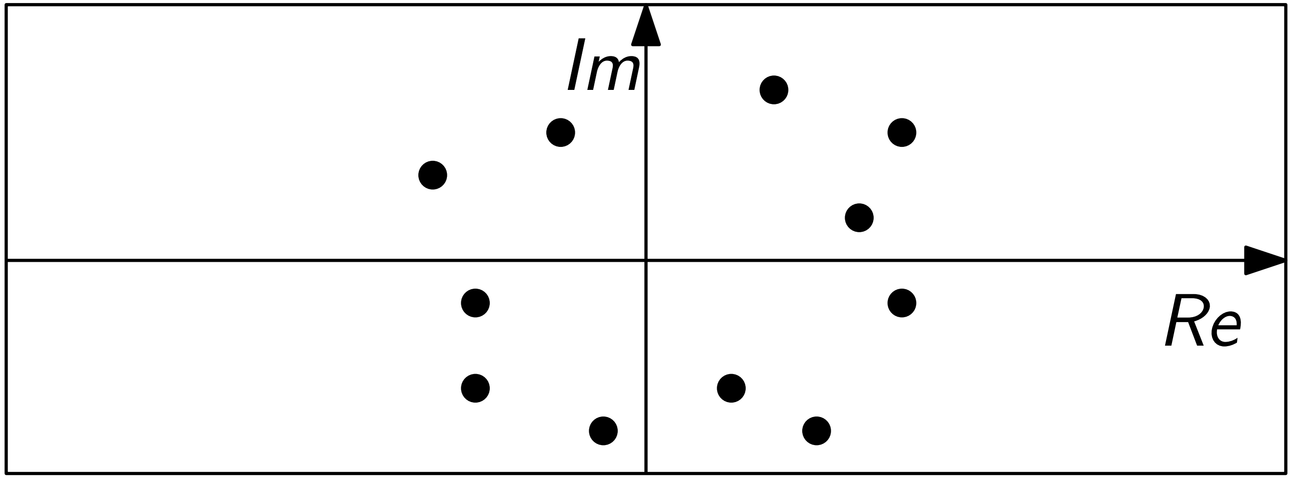
$$f(z) = f_0 + \cdots + f_d z^d \qquad f_k \in \mathbb{C}$$

Multipoint evaluation

Given d complex numbers z_k, evaluate all the $f(z_k)$.

Root finding

Find all the complex solutions ζ_k of $f(z) = 0$.



$\mathbb{C} \simeq \mathbb{R}^2$

Floating-point arithmetic

Representation

Light-year: $9\,460\,730\,472\,580\,800$ m

$$\underbrace{9.460}_{\textbf{mantissa}} \cdot 10^{\underbrace{15}_{\textbf{exponent}}} \text{ m}$$

mantissa number of digits $< m$

exponent absolute value $< \tau$

Polynomial evaluation

$$f(z) = f_0 + \cdots + f_d z^d$$

- Bit complexity : $\widetilde{O}(d(m + \log \tau))$
- Error: $O(d2^{-m})\widetilde{f}(|z|)$ where $\widetilde{f}(|z|) = \sum |f_j||z|^j$

Conditioning of root finding

$$f(z) = f_0 + \cdots + f_d z^d \qquad f_k \in \mathbb{C}$$

Condition number of a root ζ of f

Measures the displacement of ζ under an infinitesimal perturbation applied to the coefficients of f.

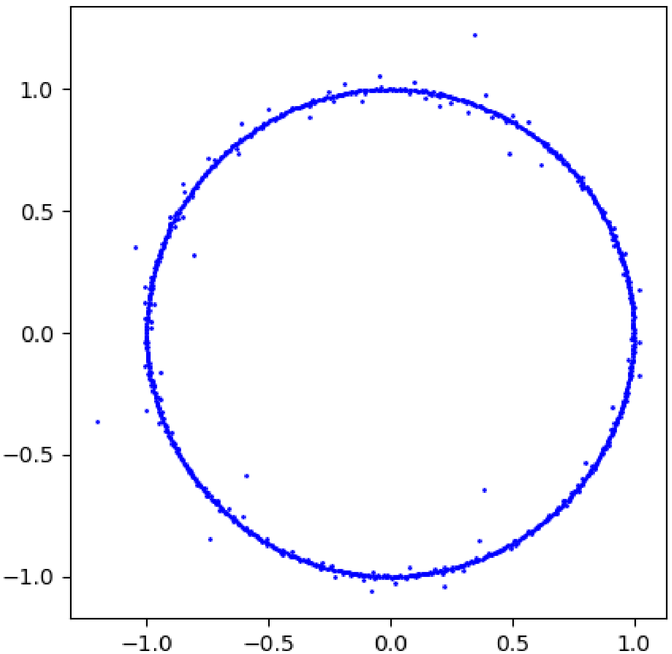
Uniform perturbation	**Relative perturbation**
$f(z) = \sum (f_j + \varepsilon_j) z^j$	$f(z) = \sum f_j (1 + \varepsilon_j) z^j$
$cond_u(f, \zeta) = \frac{(\max \lvert f_j \rvert) \sum \lvert z \rvert^j}{\lvert f'(\zeta) \rvert}$	$cond_r(f, \zeta) = \frac{\sum \lvert f_j \rvert \lvert z \rvert^j}{\lvert \zeta \rvert \lvert f'(\zeta) \rvert}$
$\kappa_u = \max cond_r(f, \zeta)$	$\kappa_r = \max cond_r(f, \zeta)$

Examples

c_j are i.i.d. Gaussian variables with mean 0

Hyperbolic

$$\sum c_j z^j$$

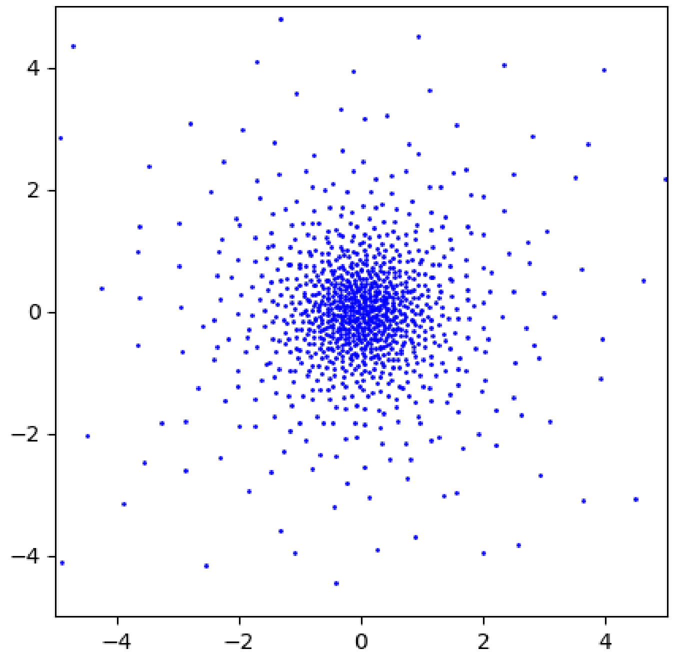


κ_u and κ_r

polynomial in d

Elliptic

$$\sum c_j \sqrt{\binom{d}{j}} z^j$$

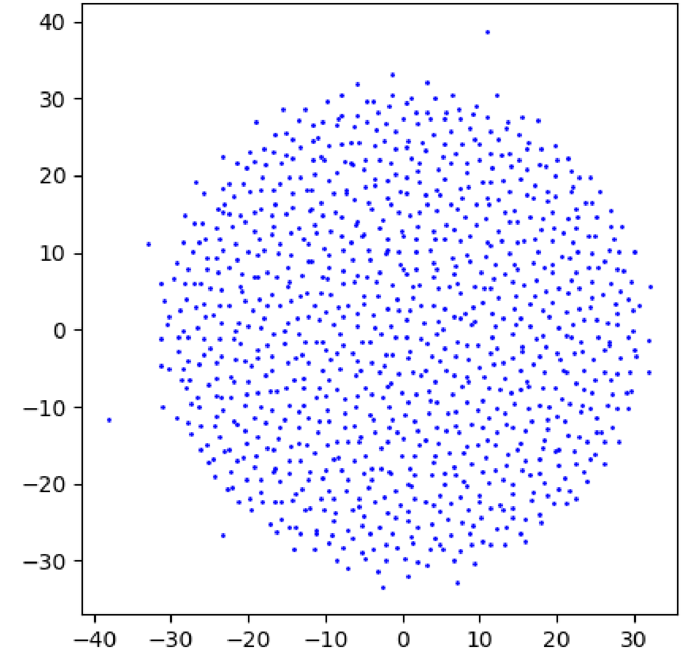


κ_u

exponential in d

Flat

$$\sum c_j \sqrt{\frac{1}{j!}} z^j$$



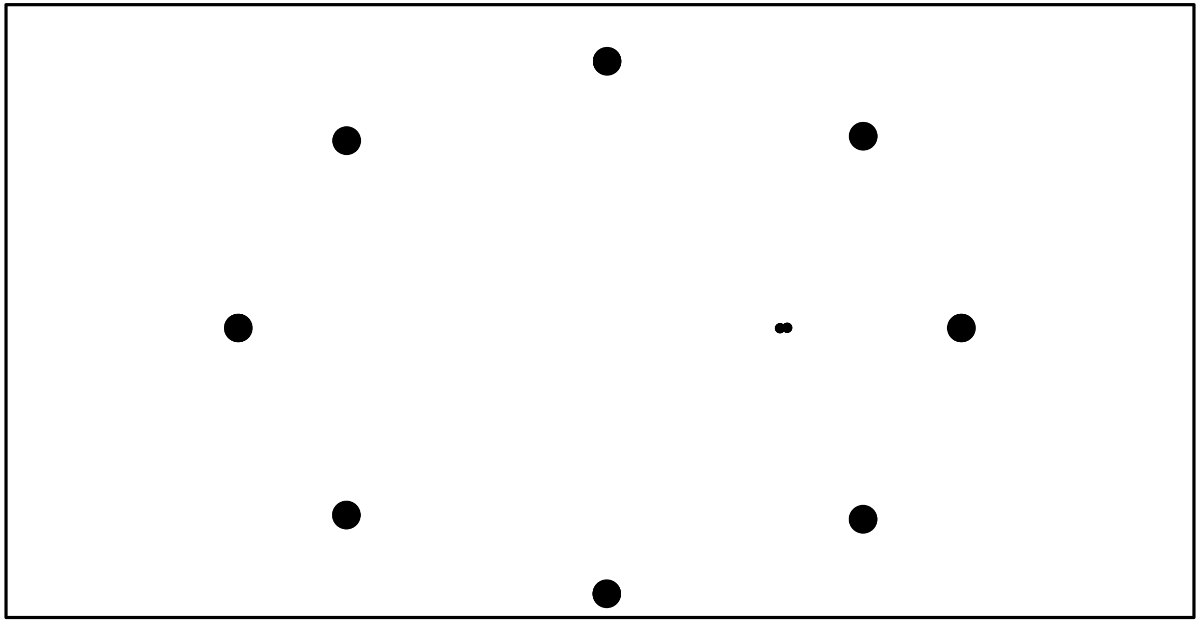
κ_u

exponential in d

Examples

Mignotte

$$z^d - 2(2z^2 - 1)^2 = 0$$

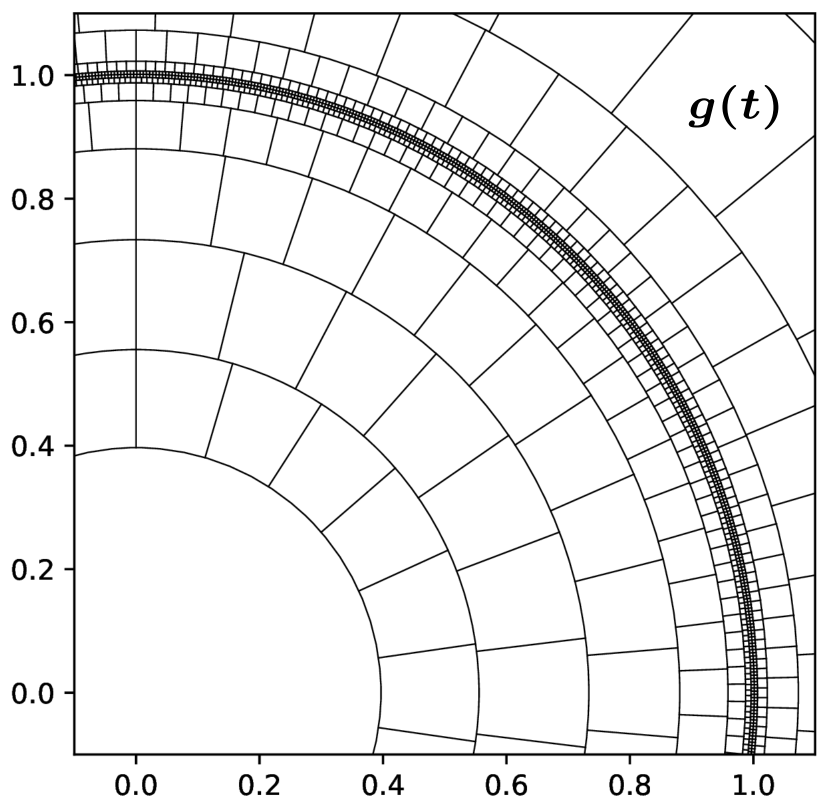


κ_r and κ_u
exponential in d

Representation of polynomials

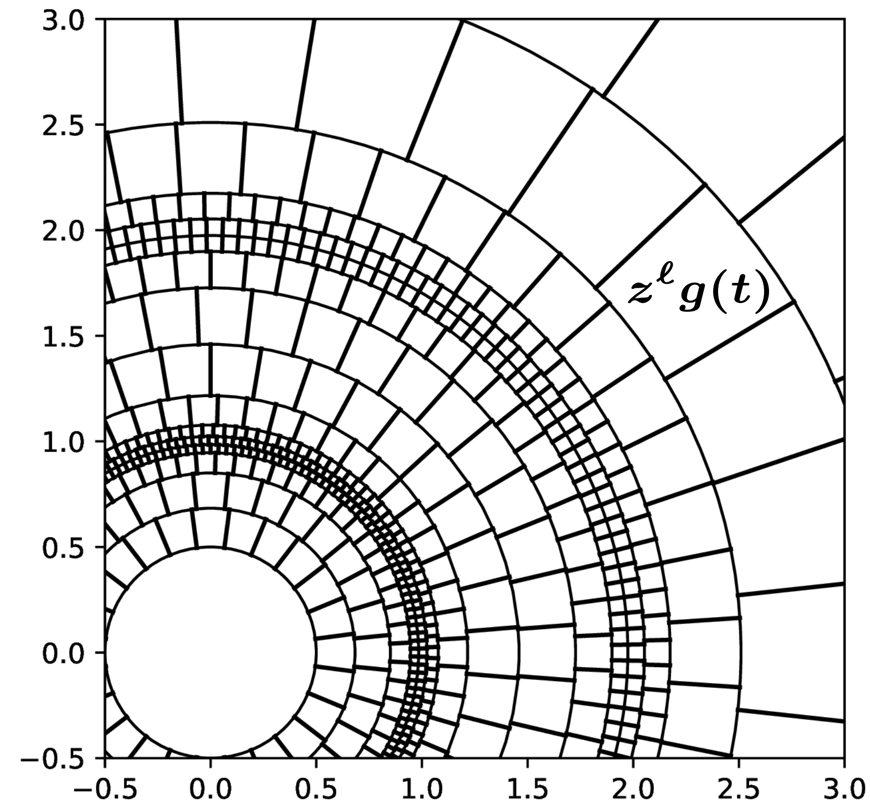
Hyperbolic approximation

Generalization of
Fixed-point representation



Relative approximation

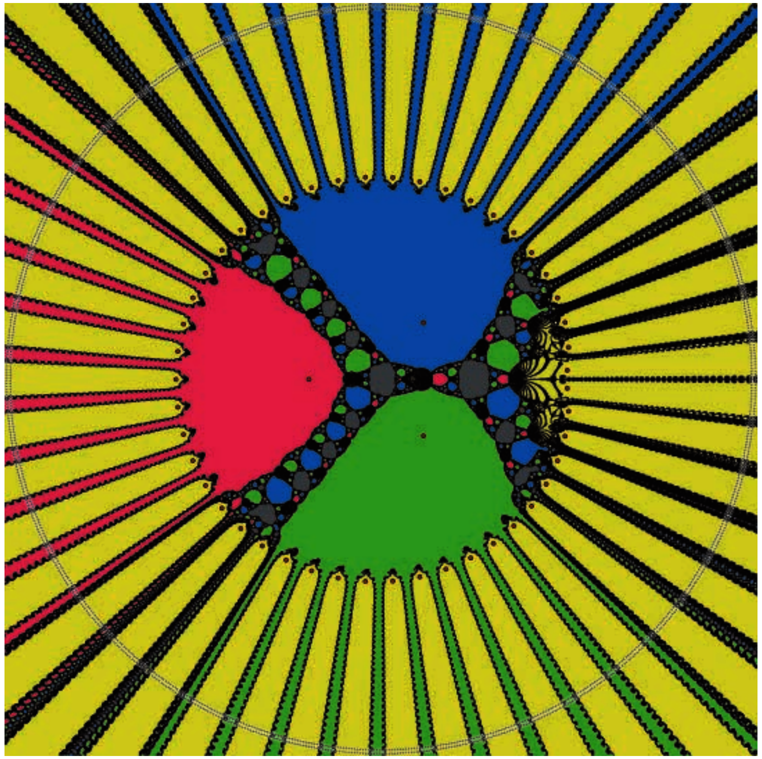
Generalization of
floating-point representation



State of the art: root finding

Newton

[Hubbard, Schleicher, Sutherland 2001]



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

Aberth-Ehrlich variant (1967)

$$F(z) = \frac{f(z)}{(z-z_2)\cdots(z-z_d)}$$

Approximate factorization

$$\|\prod(z - z_k) - f(z)\| \leq 2^{-m}\|f\|$$

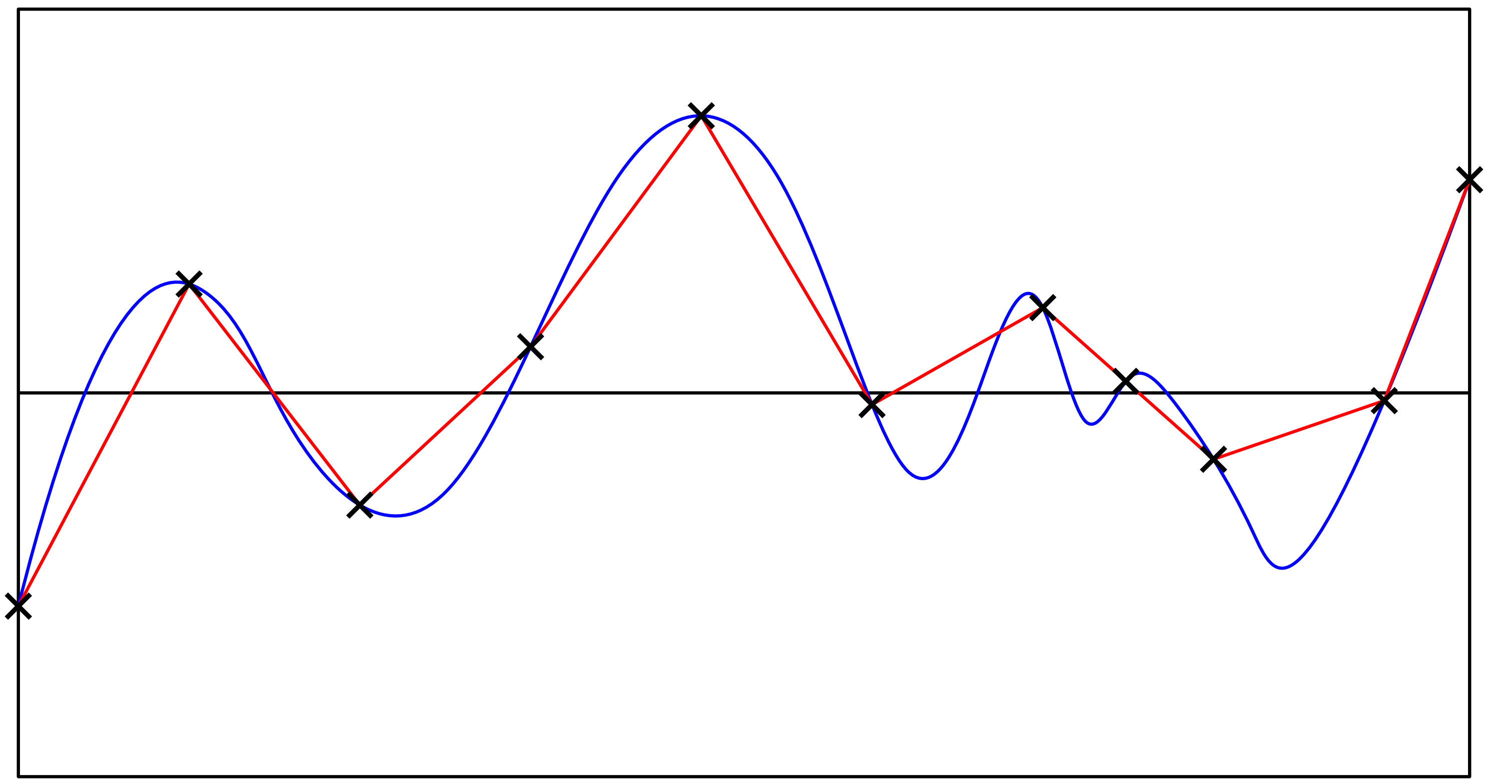
$\rightarrow$ approximation in $\widetilde{O}(d(d+m))$ bit operations
[Schönhage 1982, Pan 2002]

Other methods

Subdivision, Weierstrass, eigenvalue of companion matrix, ...

State of the art: root finding

Piecewise linear approximation



$\rightarrow$ piecewise low-degree polynomial approximations

[Cheney 1966, Powell 1982, Boyd 2006, etc.]

State of the art: multipoint evaluation

Evaluate $f(z)$ on d points with error in 2^{-m} $\quad |f_k| < 2^m$

Hörner

$$f_0 + z(f_1 + z(\cdots + z(f_{d-1} + z f_d)\cdots))$$

$\to$ multipoint evaluation in $\widetilde{O}(d^2 m)$ bit operations

Divide and conquer

$$\boxed{f(z) \mod \prod_{k=1}^{d}(z - z_k)}$$

- $\widetilde{O}(d)$ arithmetic operations [Fiduccia 1972]
- $\widetilde{O}(d(d+m))$ bit operations [van der Hoeven 2008]

Piecewise low-degree polynomial approximation

$\to$ multipoint evaluation in:

- $\widetilde{O}(d^{3/2} m^{3/2})$ bit operations [van der Hoeven 2008]

State of the art: multipoint evaluation

Evaluate $f(z)$ on d points with error in 2^{-m} $\quad |f_k| < 2^m$

Hörner

$$f_0 + z(f_1 + z(\cdots + z(f_{d-1} + z f_d)\cdots))$$

$\rightarrow$ multipoint evaluation in $\widetilde{O}(d^2 m)$ bit operations

Divide and conquer

$\boxed{f(z) \mod \prod_{k=1}^{d/2}(z - z_k)}$ $\quad$ $\boxed{f(z) \mod \prod_{k=d/2}^{d}(z - z_k)}$

- $\widetilde{O}(d)$ arithmetic operations [Fiduccia 1972]
- $\widetilde{O}(d(d+m))$ bit operations [van der Hoeven 2008]

Piecewise low-degree polynomial approximation

$\rightarrow$ multipoint evaluation in:

- $\widetilde{O}(d^{3/2} m^{3/2})$ bit operations [van der Hoeven 2008]

State of the art: multipoint evaluation

Evaluate $f(z)$ on d points with error in 2^{-m} $\quad |f_k| < 2^m$

Hörner

$$f_0 + z(f_1 + z(\cdots + z(f_{d-1} + z f_d)\cdots))$$

$\rightarrow$ multipoint evaluation in $\widetilde{O}(d^2 m)$ bit operations

Divide and conquer

$\boxed{f(z) \mod (z - z_1)}$ $\cdots$ $\boxed{f(z) \mod (z - z_k)}$ $\cdots$ $\boxed{f(z) \mod (z - z_d)}$

- $\widetilde{O}(d)$ arithmetic operations [Fiduccia 1972]
- $\widetilde{O}(d(d+m))$ bit operations [van der Hoeven 2008]

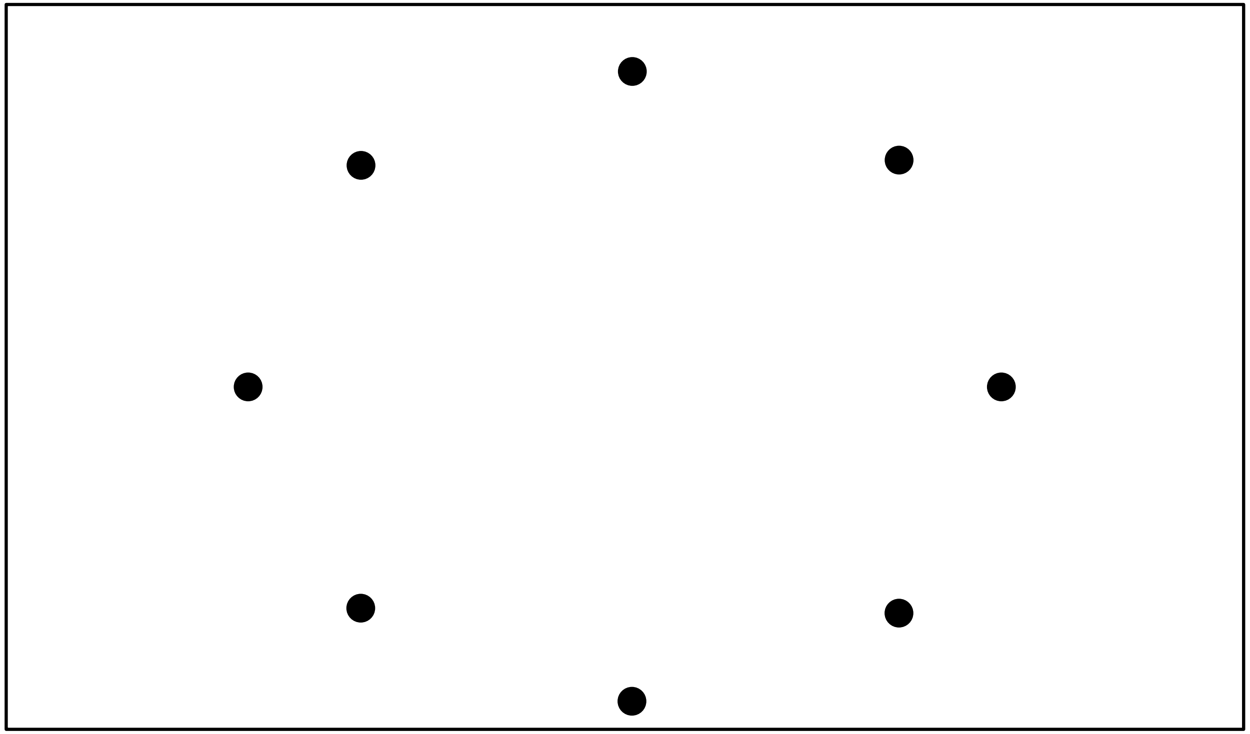
Piecewise low-degree polynomial approximation

$\rightarrow$ multipoint evaluation in:

- $\widetilde{O}(d^{3/2} m^{3/2})$ bit operations [van der Hoeven 2008]

State of the art: multipoint evaluation

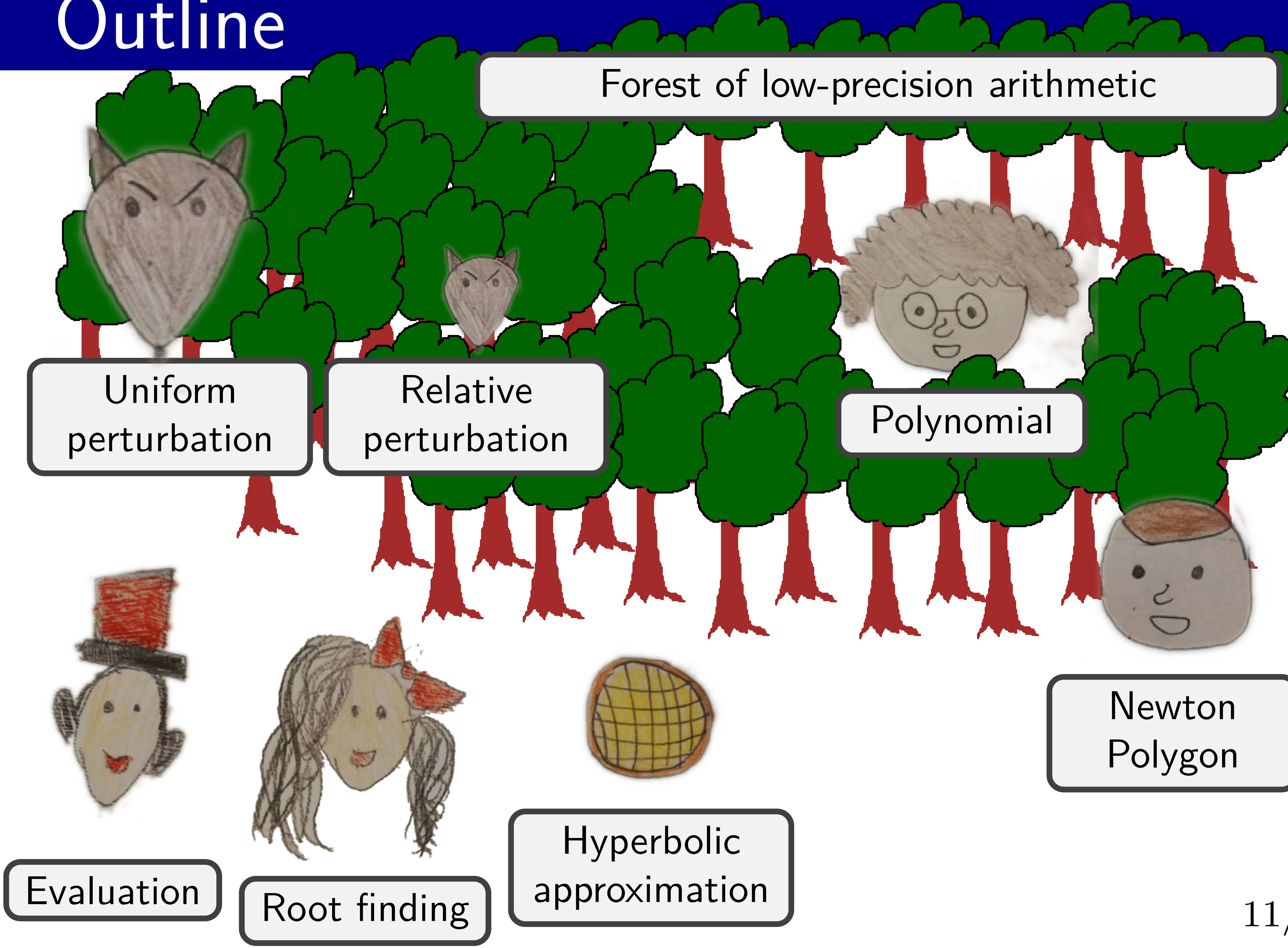
Evaluation on the roots of unity $w_k = e^{i\pi k/d}$



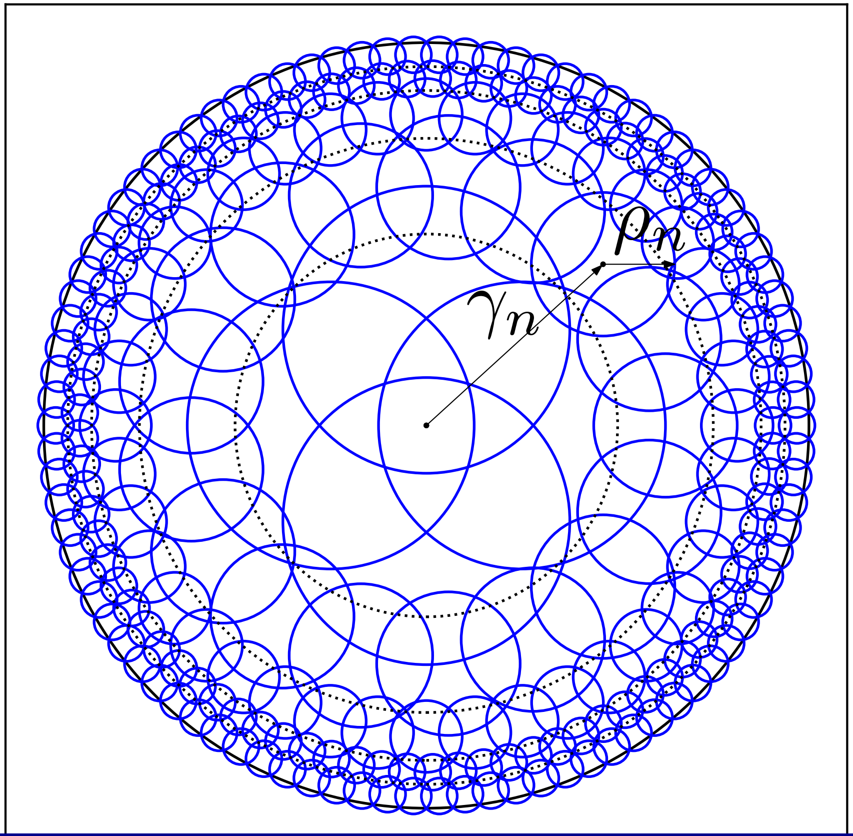
$\rightarrow$ evaluation on w_k in $\widetilde{O}(dm)$ using Fast Fourier Transform [Gauss 1805, Cooley, Tukey 1965, Schönhage 1982]

$\rightarrow$ interpolation from $f(w_k)$ in $\widetilde{O}(dm)$

Outline


Forest of low-precision arithmetic
Uniform perturbation
Relative perturbation
Polynomial
Newton Polygon
Evaluation
Root finding
Hyperbolic approximation

Hyperbolic approximation



$$0 \le n < N - 1 = O\left(\log \frac{d}{m}\right)$$

$$\begin{cases} \gamma_n = 1 - \dfrac{3}{4}\dfrac{1}{2^n} \\ \rho_n = \dfrac{3}{8}\dfrac{1}{2^n} \end{cases}$$

$$g(z) = f(\gamma + \rho z) \mod z^m$$

$\widetilde{O}(\frac{d}{m})$ polynomials of degree m

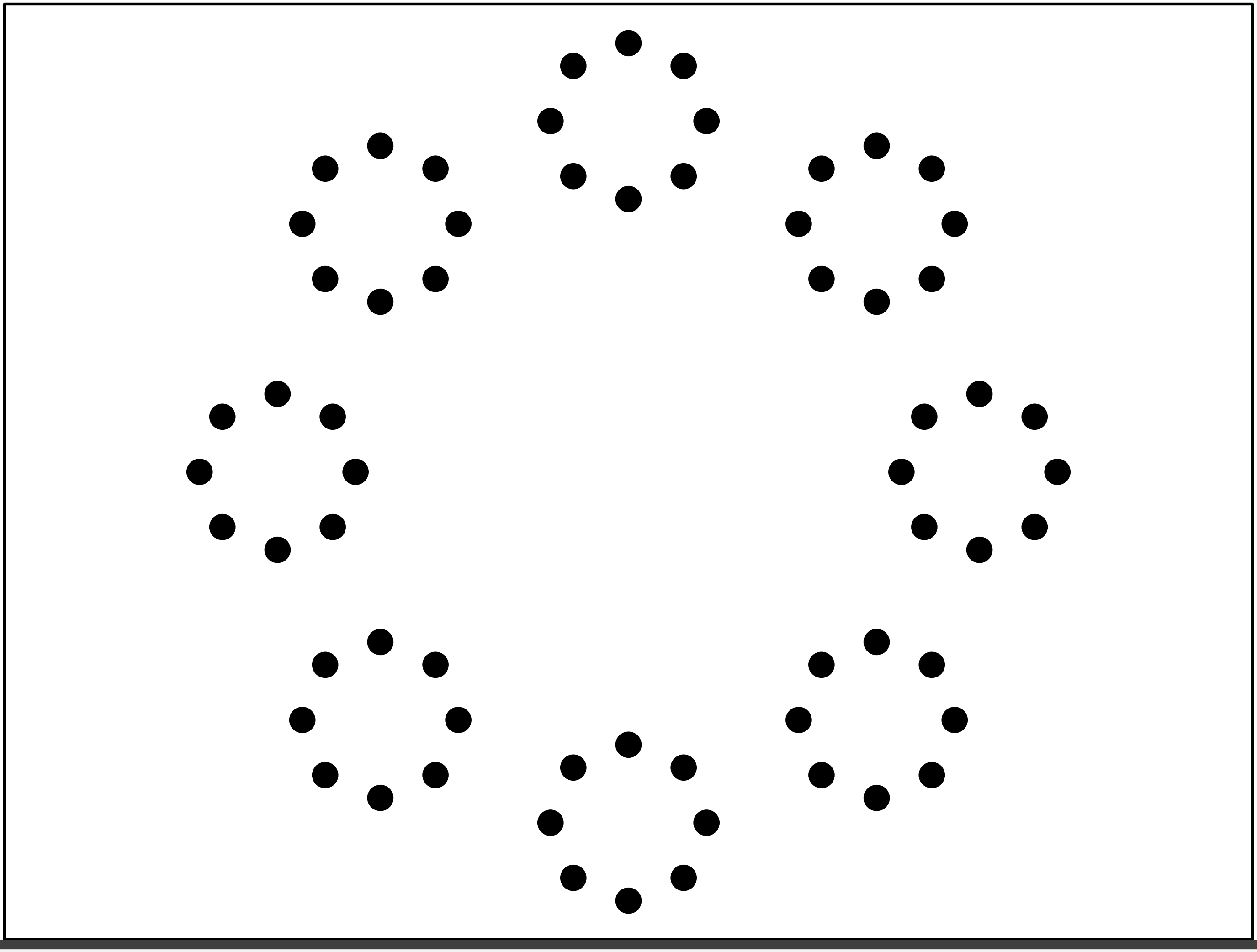
Theorem (hyperbolic approximation) [M. 2021]

It is possible to compute all g of degree m satisfying

$$\|f(\gamma + \rho z) - g(z)\| \le 2^{-m}\|f\|$$

in $\widetilde{O}(d(m+\tau))$ bit operations

Hyperbolic approximation computation

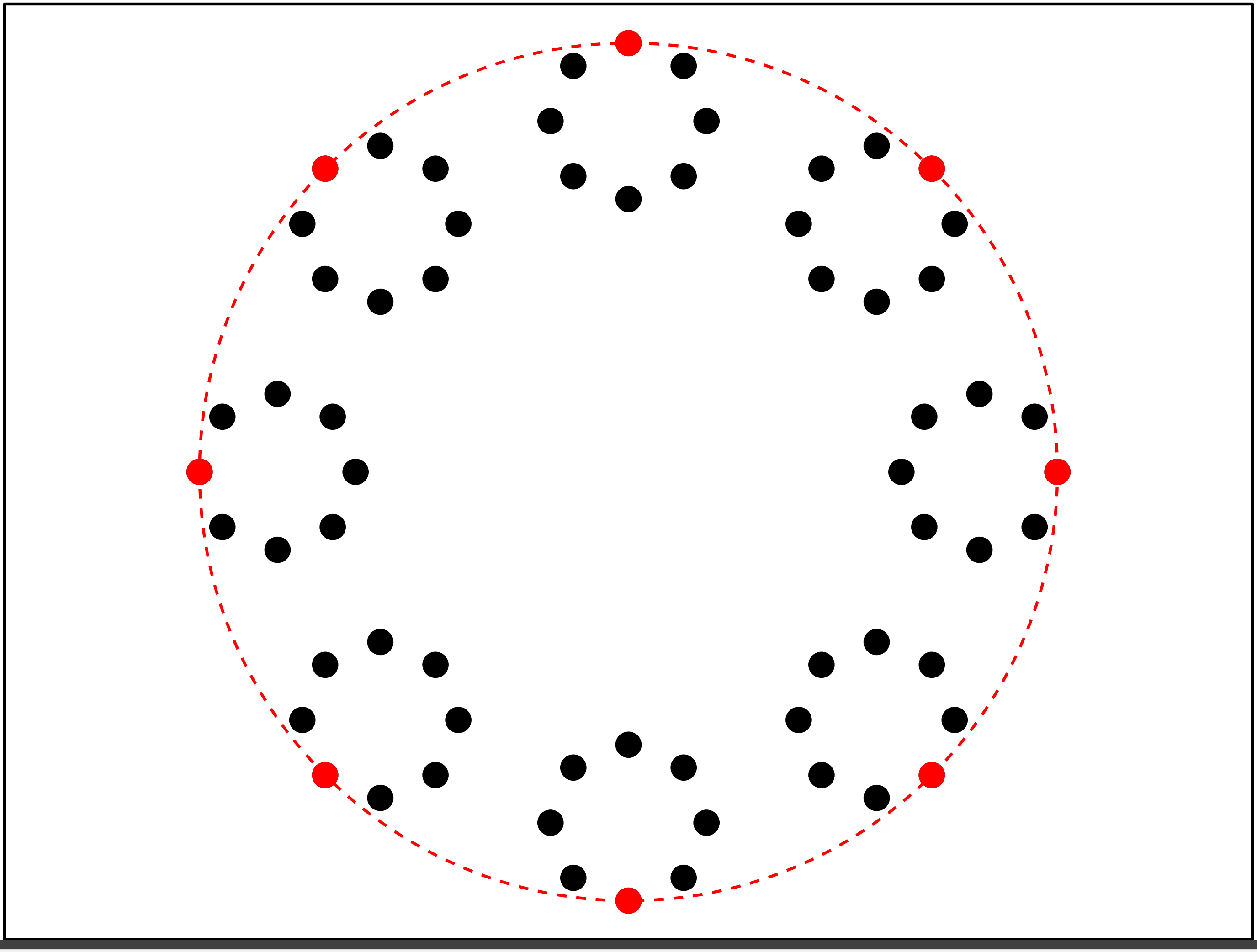


Do m times

$\quad$ Scaling d coefficients

$\quad$ FFT on d/m roots of unity

Hyperbolic approximation computation

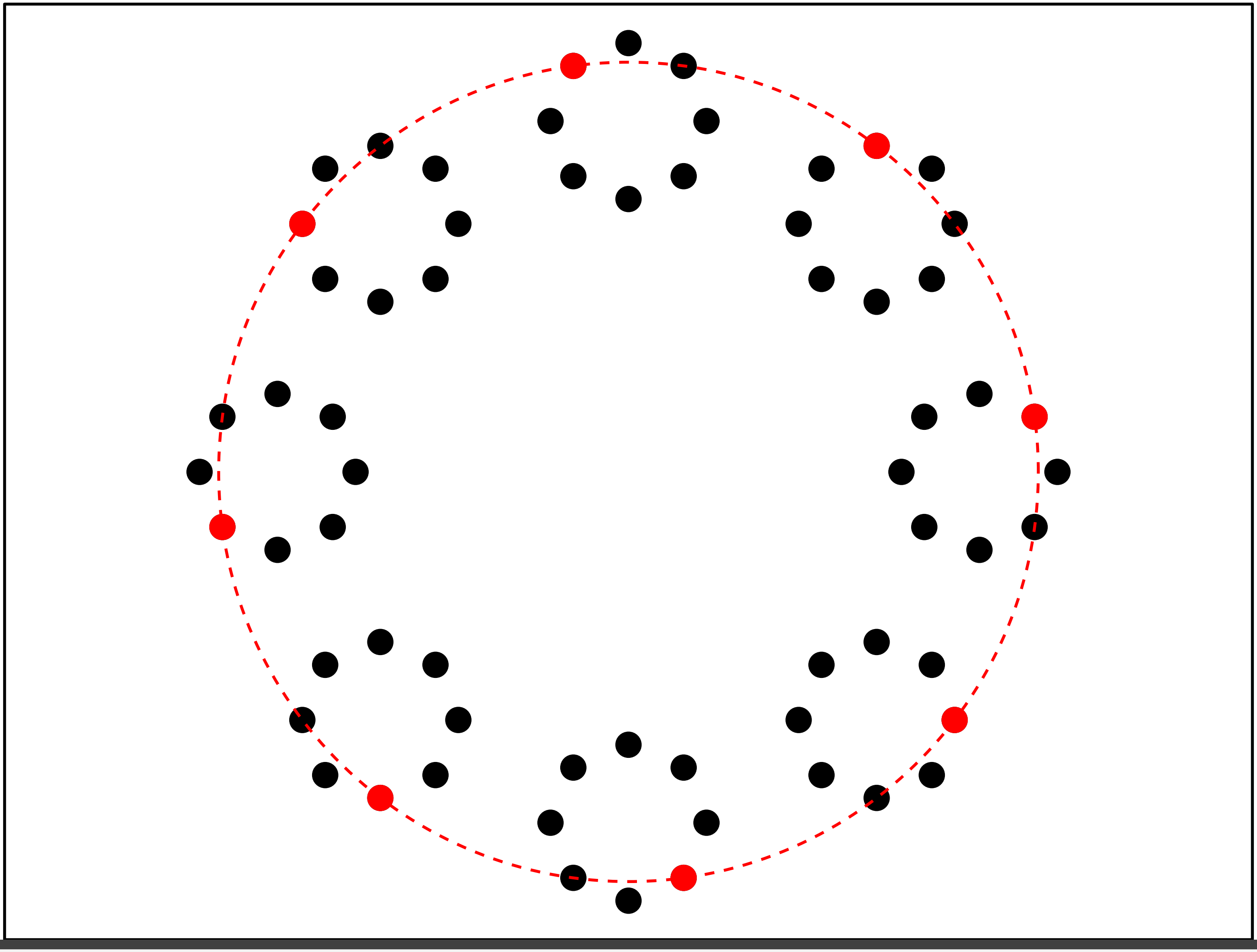


Do m times

$\quad$ SCALING d coefficients

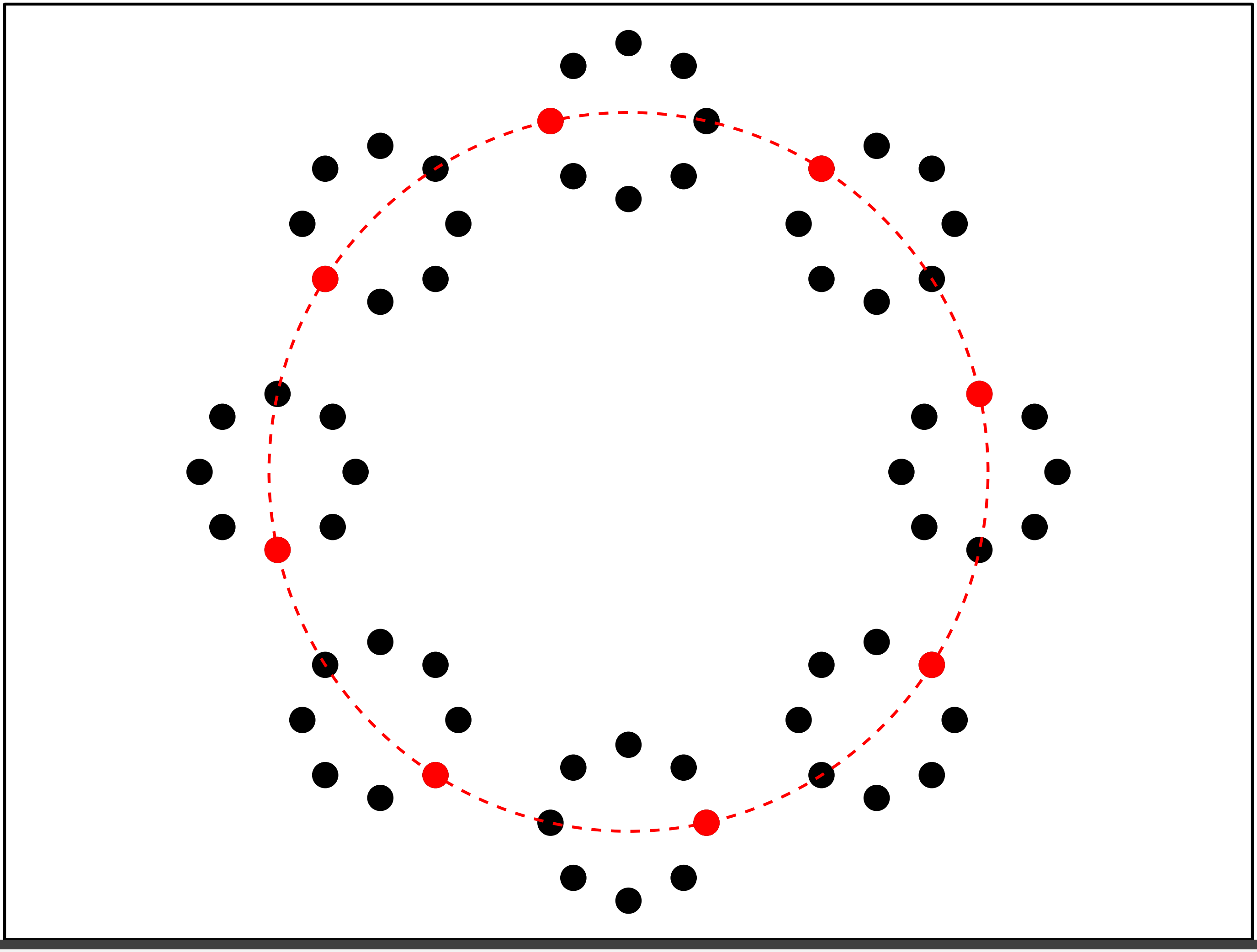
$\quad$ FFT on d/m roots of unity

Hyperbolic approximation computation



Do m times
    Scaling d coefficients
    FFT on d/m roots of unity

Hyperbolic approximation computation

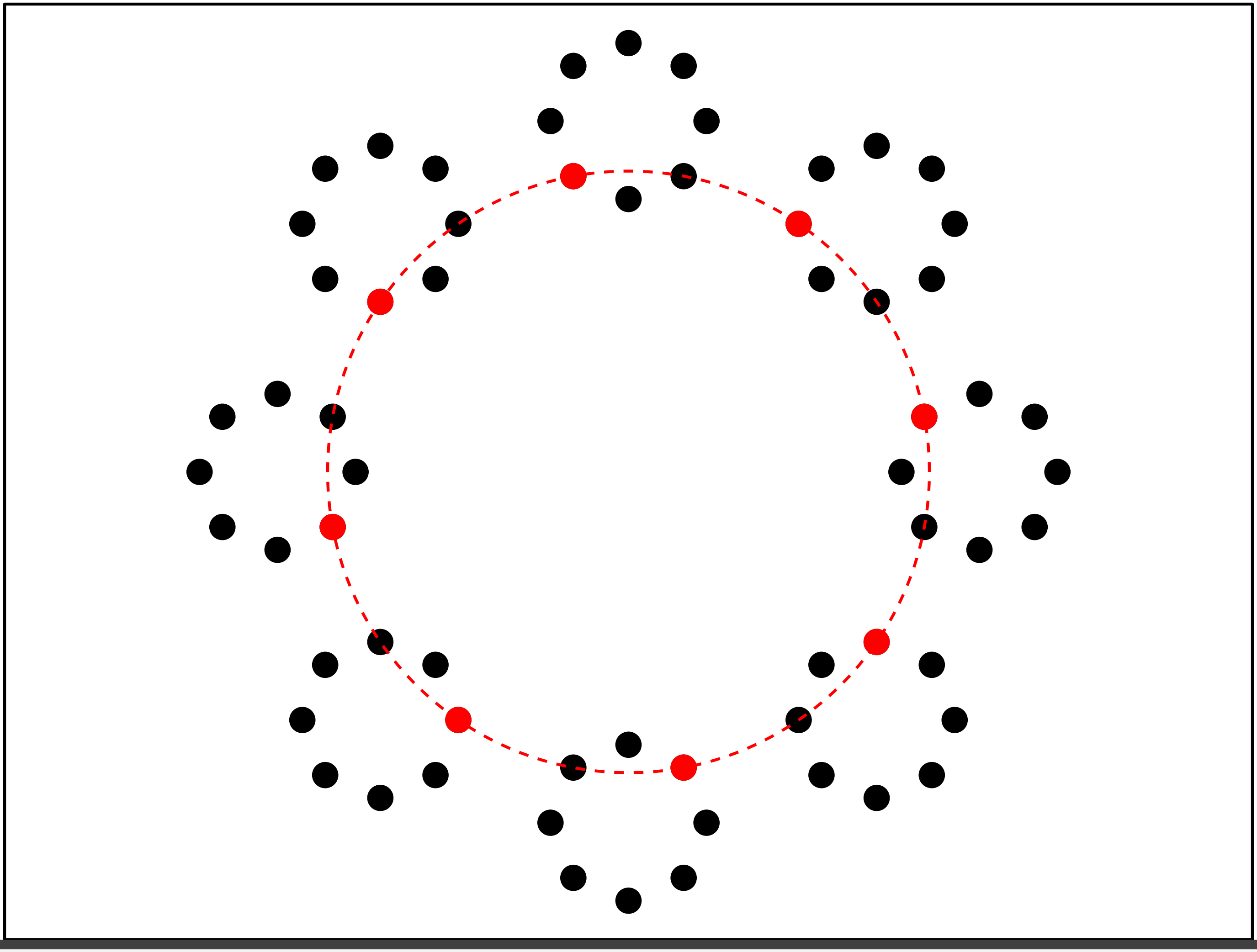


Do m times

SCALING d coefficients

FFT on d/m roots of unity

Hyperbolic approximation computation

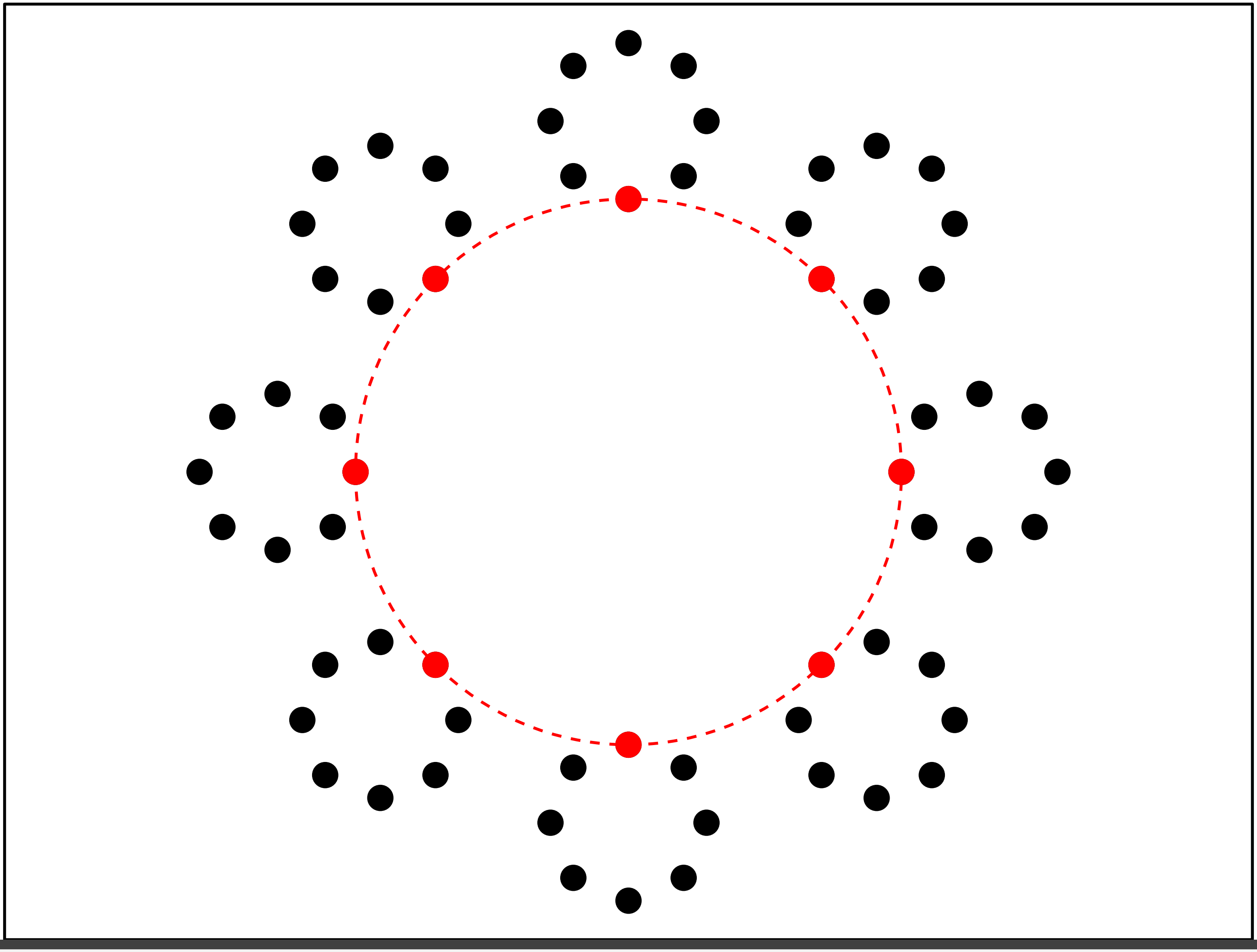


Do m times

$\quad$ SCALING d coefficients

$\quad$ FFT on d/m roots of unity

Hyperbolic approximation computation

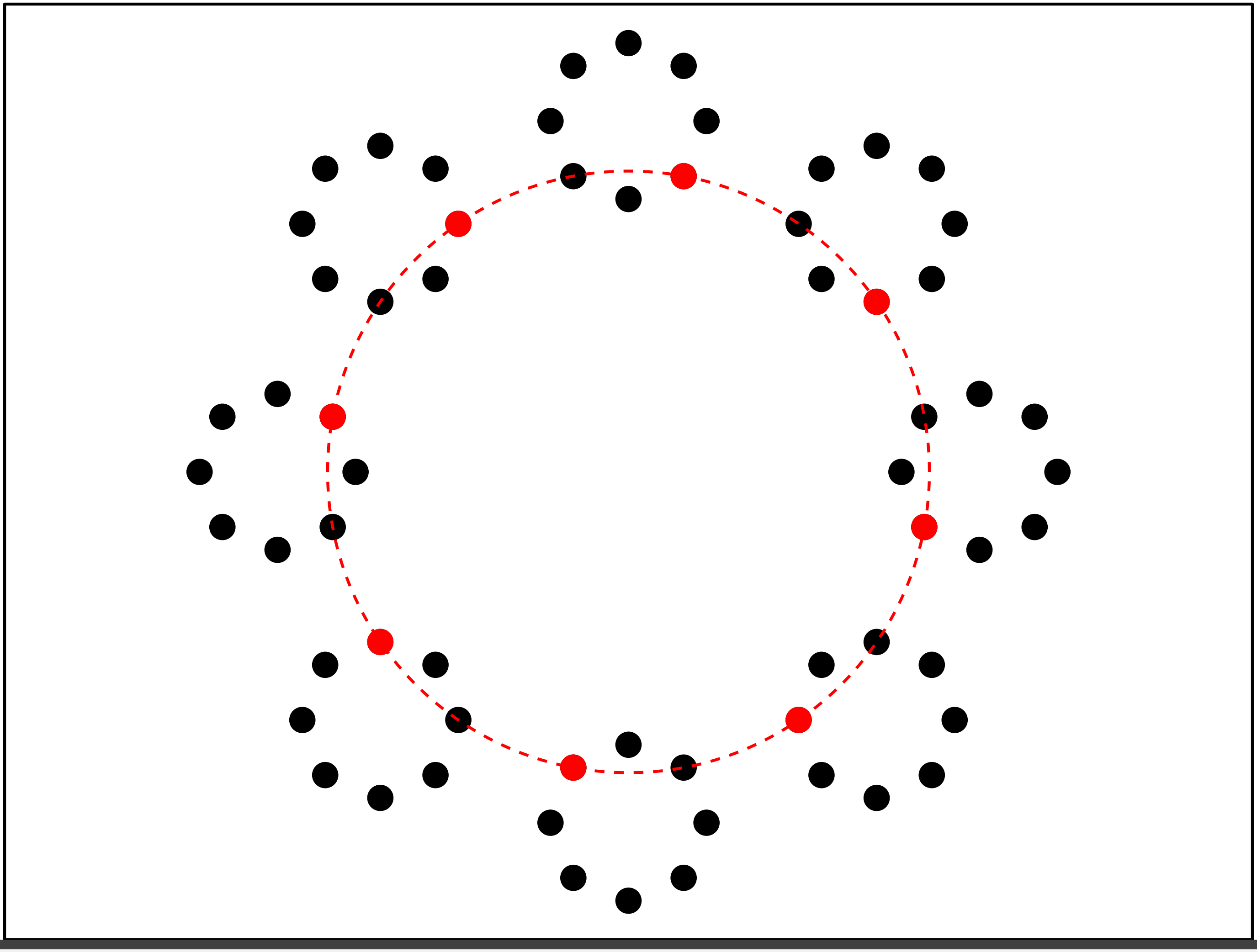


Do m times

$\quad$ Scaling d coefficients

$\quad$ FFT on d/m roots of unity

Hyperbolic approximation computation

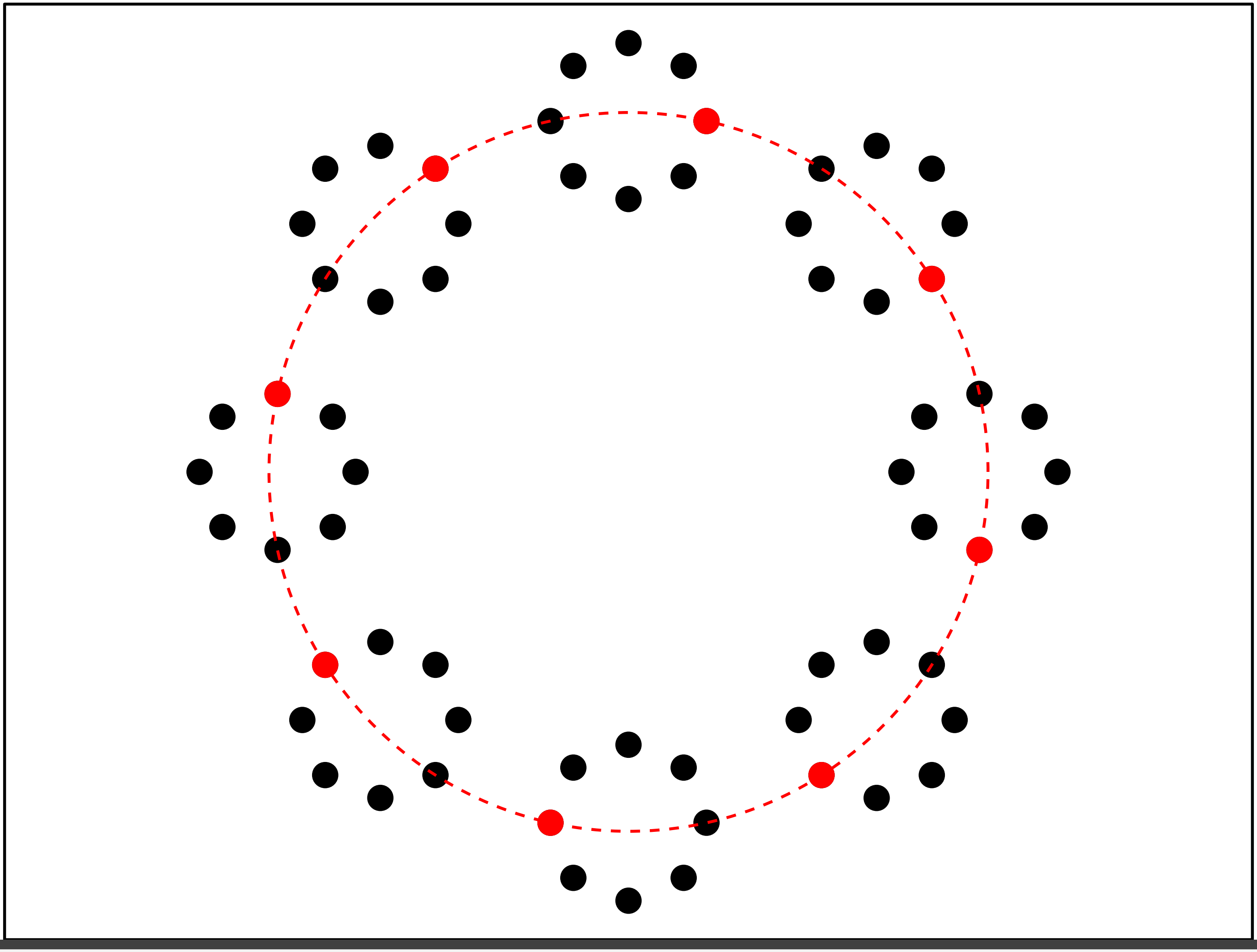


Do m times

$\quad$ $\textsc{Scaling}$ d coefficients

$\quad$ FFT on d/m roots of unity

Hyperbolic approximation computation

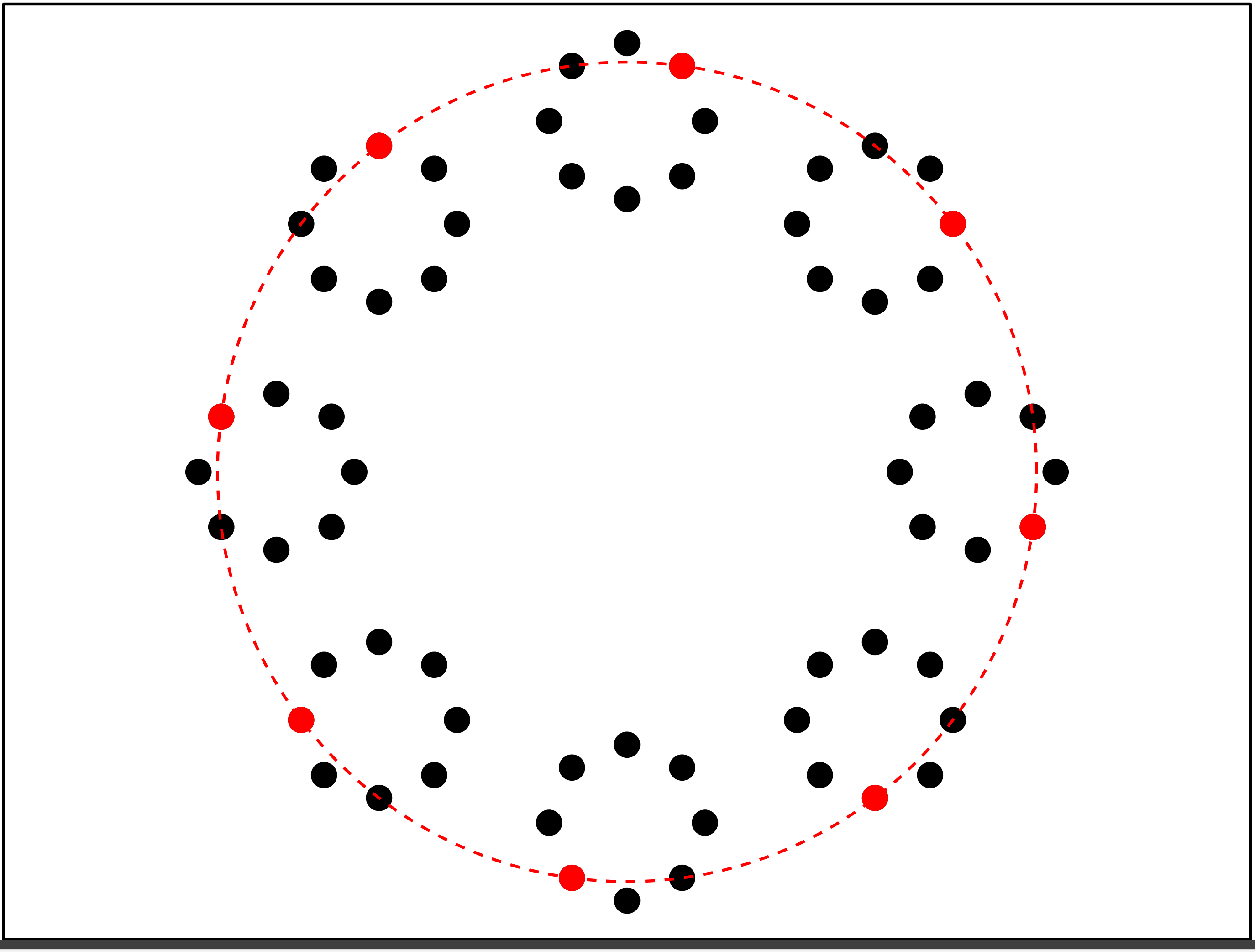


Do m times

$\quad$ Scaling d coefficients

$\quad$ FFT on d/m roots of unity

Hyperbolic approximation computation

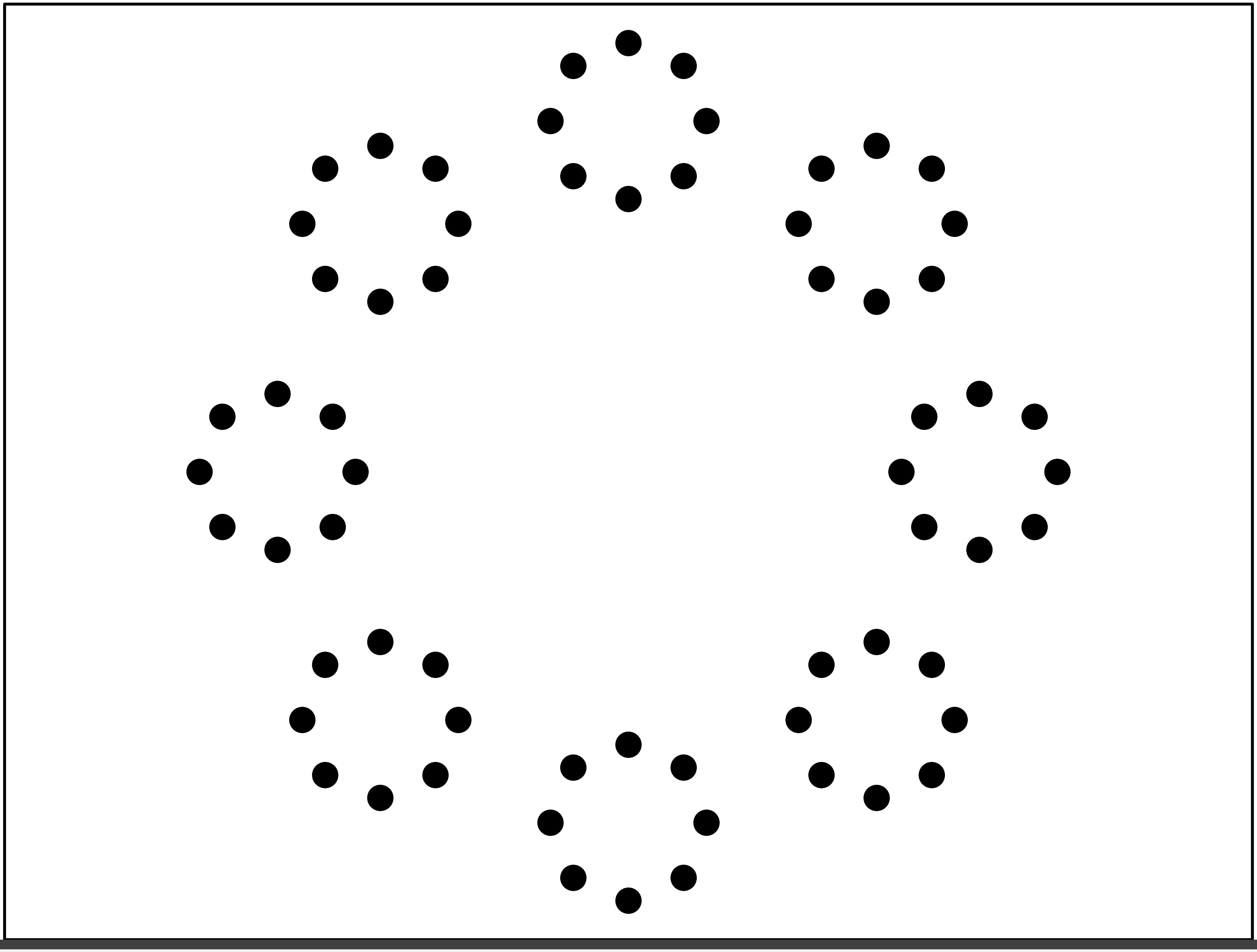


Do m times

$\textsc{Scaling}$ d coefficients

FFT on d/m roots of unity

Hyperbolic approximation computation

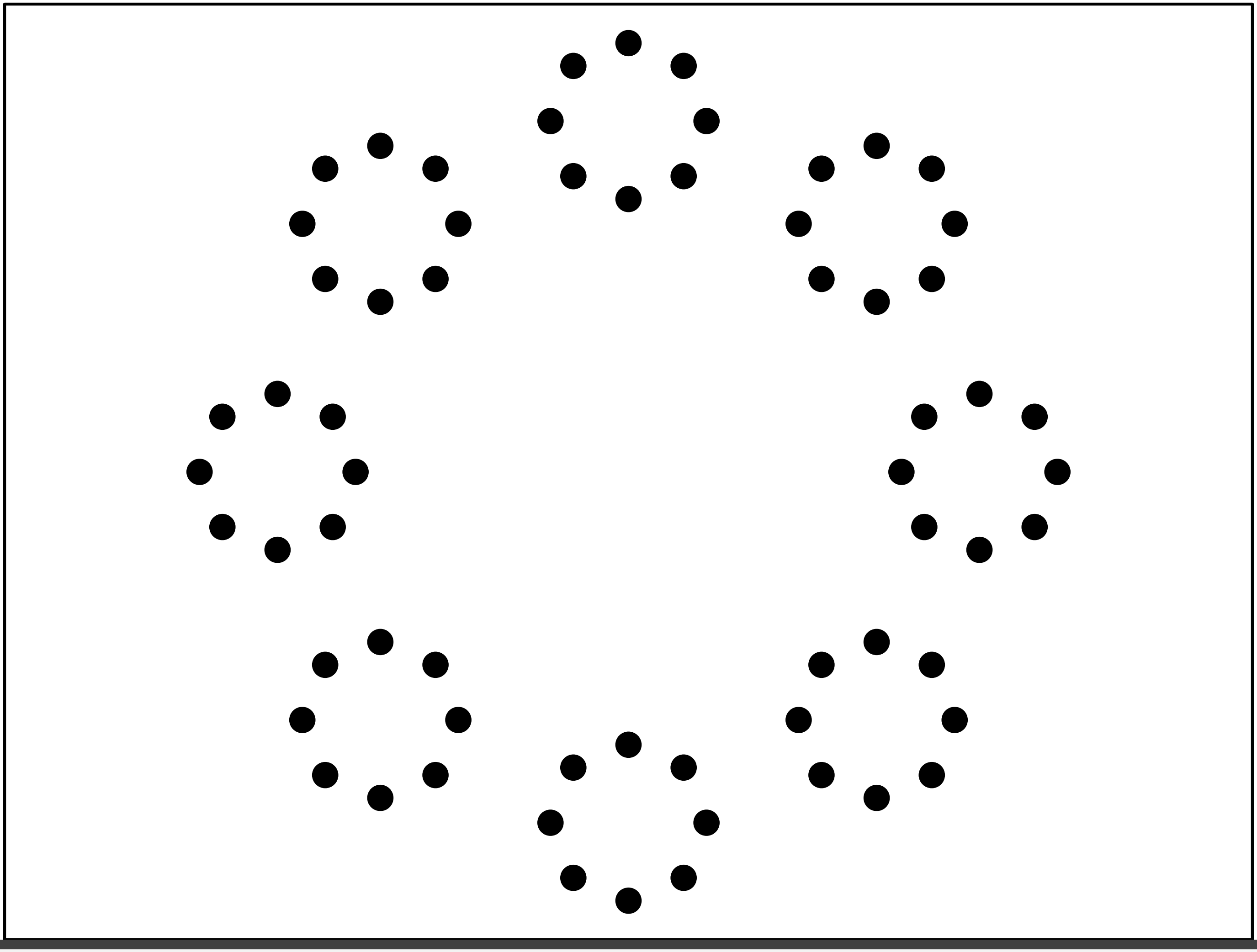


Do m times

$\textsc{Scaling}$ d coefficients $\widetilde{O}(dm)$

FFT on d/m roots of unity $\widetilde{O}(d)$

Hyperbolic approximation computation

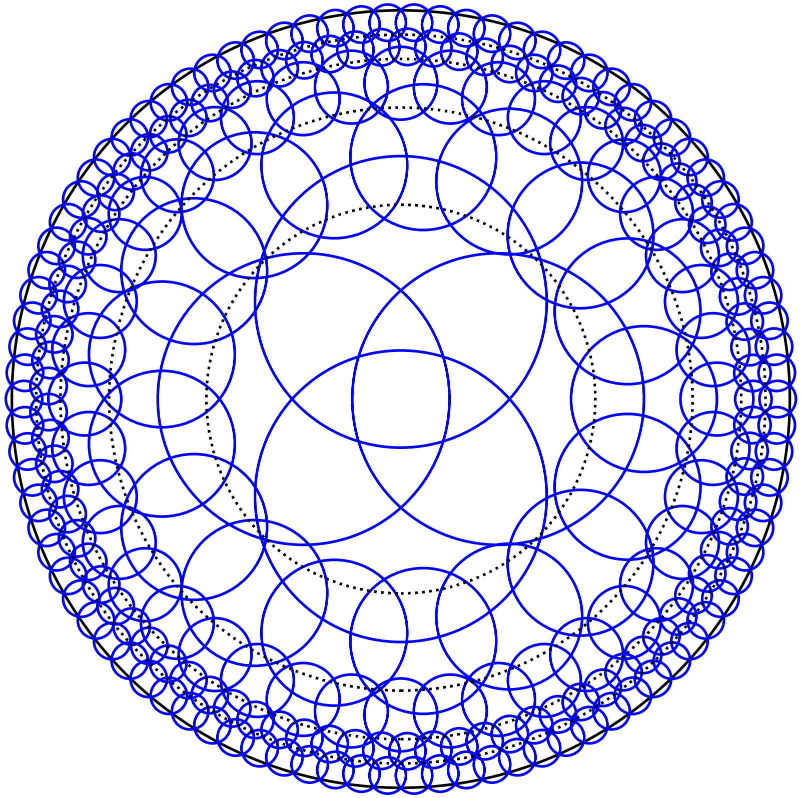


Do m times

$\qquad$ SCALING d coefficients $\qquad \widetilde{O}(d)$

$\qquad$ FFT on d/m roots of unity $\qquad \widetilde{O}(d)$

Multipoint evaluation in $\widetilde{O}(d(m+\tau))$



INPUT:

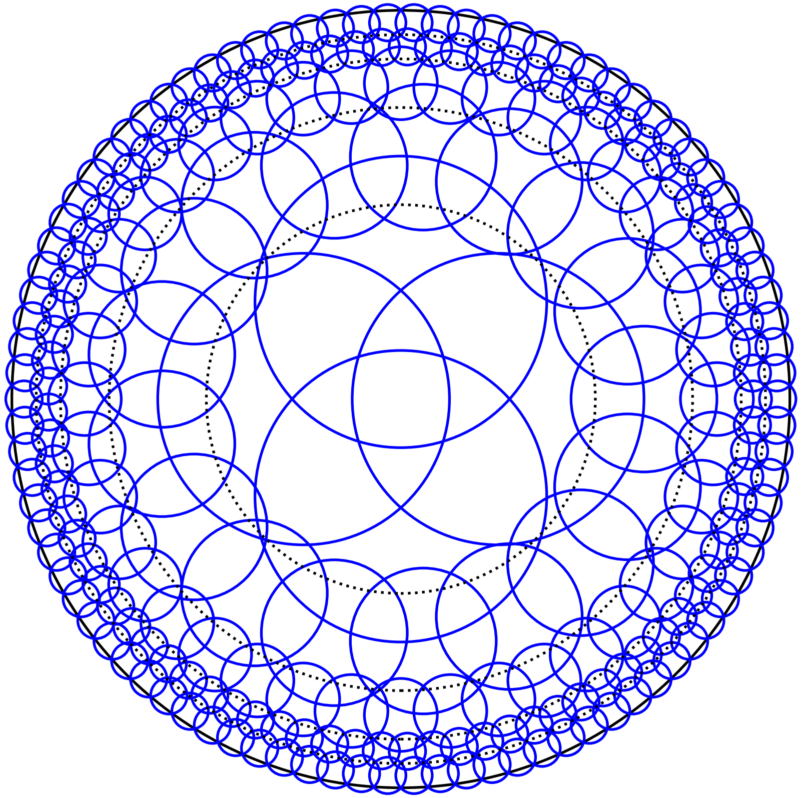
- f polynomial of degree d
- d points z_k
- precision m

OUTPUT:

- y_k such that
  $|y_k - f(z_k)| < 2^{-m}\|f\|$

Compute m-hyperbolic approximation of f	$\widetilde{O}(d(m+\tau))$
For each pair of disk D and polynomial g:	$O(d/m)$
QUERY the n_k points in D	$\widetilde{O}(n_k)$
EVALUATE g on the n_k points	$\widetilde{O}(m(n_k+m))$

Root finding in $\widetilde{O}(d(\tau + \log \kappa_u))$



INPUT:

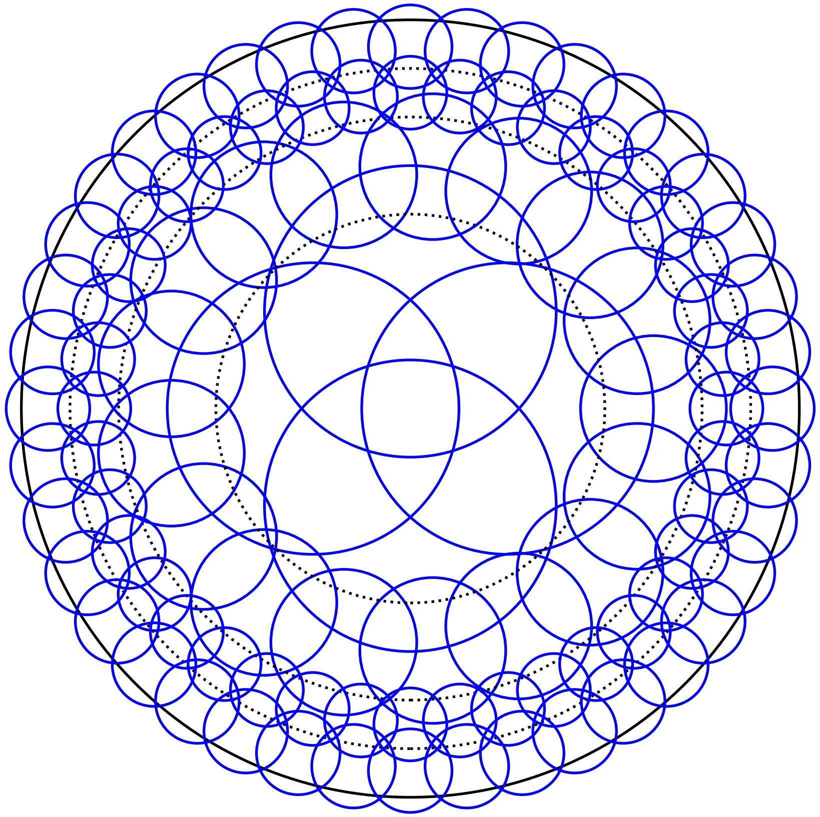
- f squarefree polynomial

OUTPUT:

- d root-isolating disks

Compute 1-hyperbolic approximation of f	$\widetilde{O}(d)$
For each polynomial g:	$O(d)$
APPROXIMATE roots of g	$\widetilde{O}(1)$
COMPUTE enclosing disks	$\widetilde{O}(1)$
Check if we have d isolating disks	$\widetilde{O}(d)$

Root finding in $\widetilde{O}(d(\tau + \log \kappa_u))$



Input:

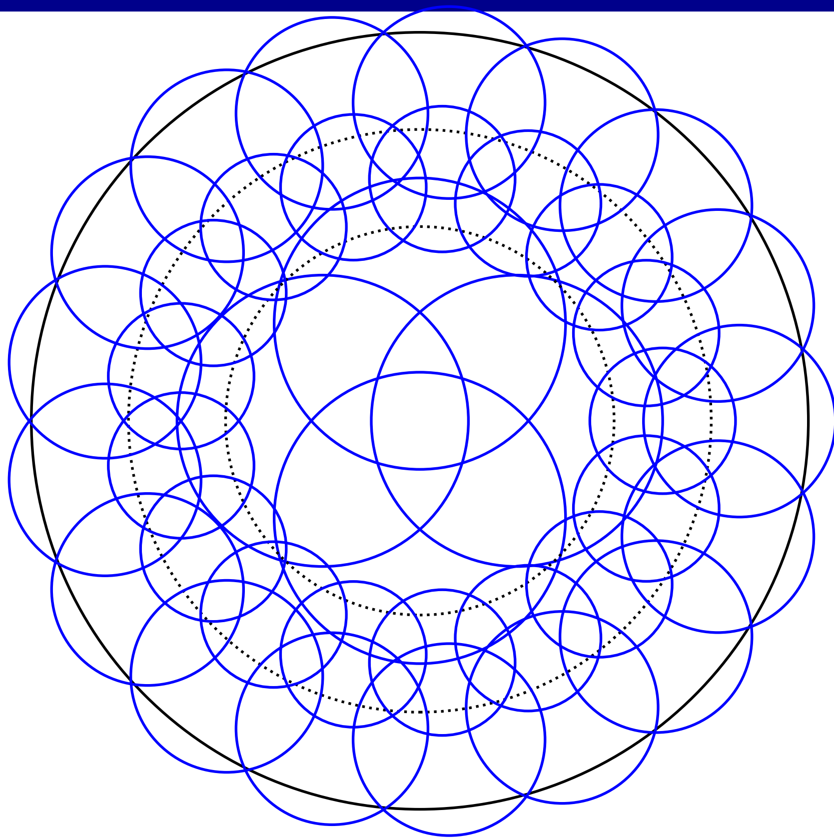
- f squarefree polynomial

Output:

- d root-isolating disks

Compute 2-hyperbolic approximation of f	$\widetilde{O}(d)$
For each polynomial g:	$O(d)$
Approximate roots of g	$\widetilde{O}(1)$
Compute enclosing disks	$\widetilde{O}(1)$
Check if we have d isolating disks	$\widetilde{O}(d)$

Root finding in $\widetilde{O}(d(\tau + \log \kappa_u))$



INPUT:

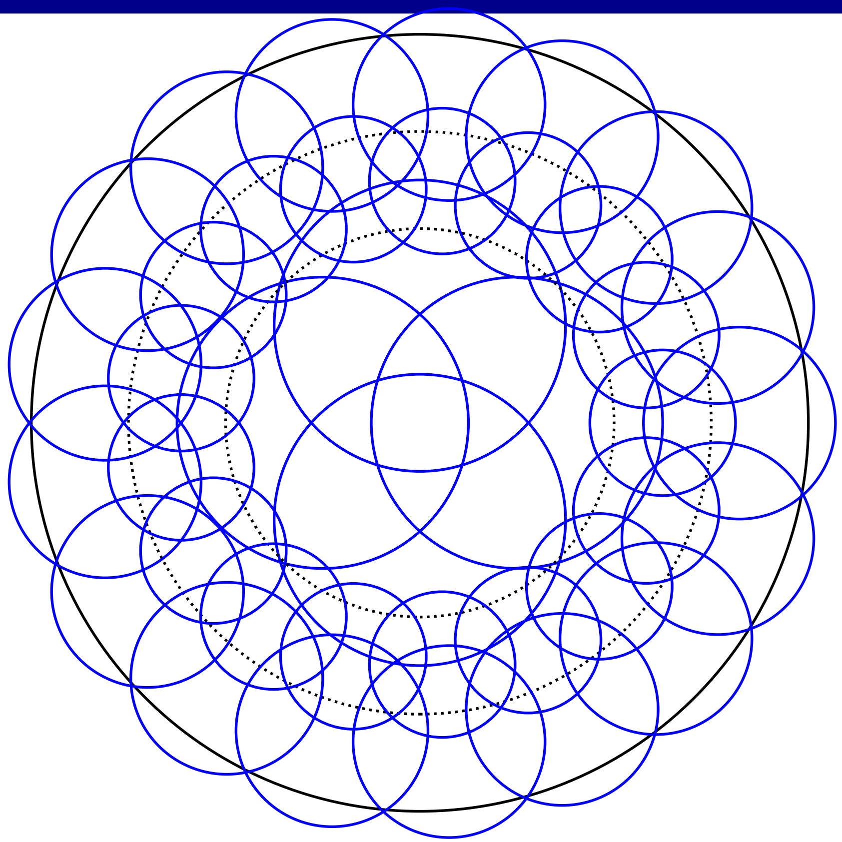
- f squarefree polynomial

OUTPUT:

- d root-isolating disks

Compute m-hyperbolic approximation of f	$\widetilde{O}(d(m+\tau))$
For each polynomial g:	$O(d/m)$
APPROXIMATE roots of g	$\widetilde{O}(m^2)$
COMPUTE enclosing disks	$\widetilde{O}(m^2)$
Check if we have d isolating disks	$\widetilde{O}(dm)$

Root finding in $\widetilde{O}(d(\tau + \log \kappa_u))$



INPUT:

- f squarefree polynomial

OUTPUT:

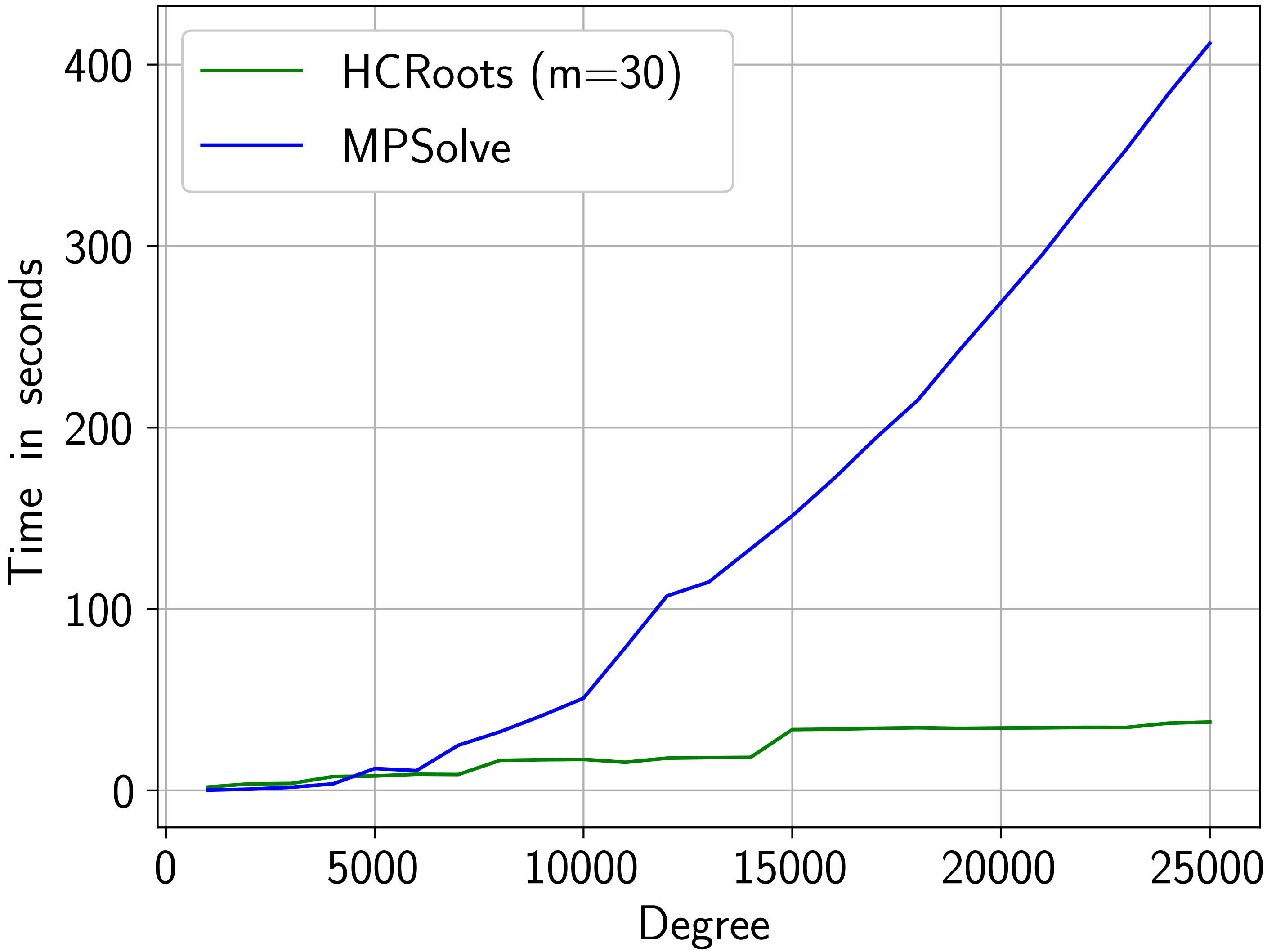
- d root-isolating disks

COMPLEXITY:

- $\widetilde{O}(d(m+\tau))$ bit operations
- m in $\widetilde{O}\left(\log(\kappa_u)\right)$

Compute m-hyperbolic approximation of f	$\widetilde{O}(d(m+\tau))$
For each polynomial g:	$O(d/m)$
APPROXIMATE roots of g	$\widetilde{O}(m^2)$
COMPUTE enclosing disks	$\widetilde{O}(m^2)$
Check if we have d isolating disks	$\widetilde{O}(dm)$

Root finding in $\widetilde{O}(d(\tau + \log \kappa_u))$

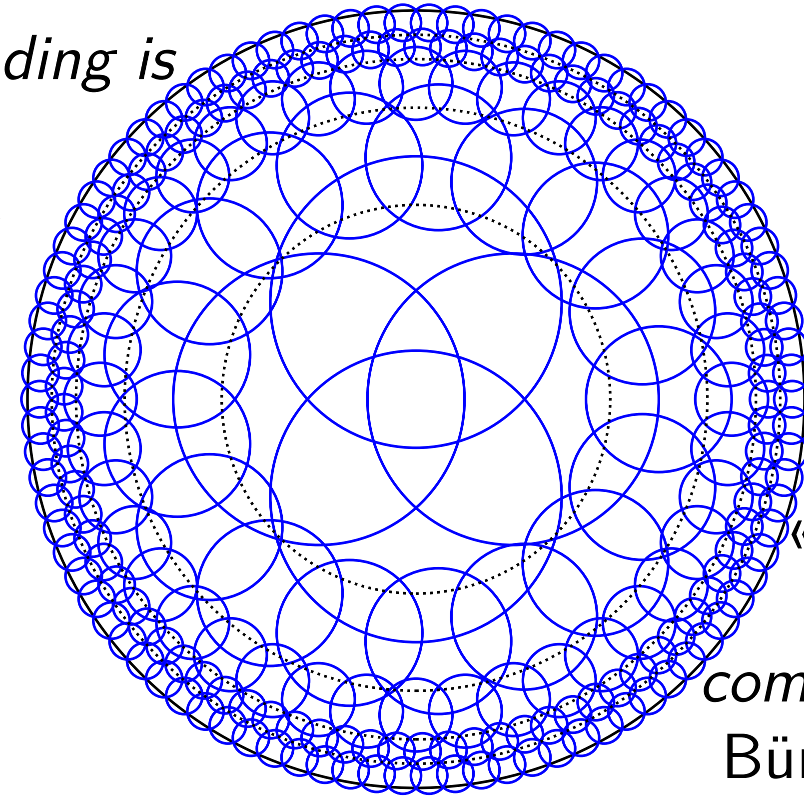


Root finding with small source code

```python
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 2 of the License, or
# (at your option) any later version.

import numpy as np

# Compute the disks of a hyperbolic covering
def disks(d, m):
    N = np.math.ceil(np.log2(3*np.e*d/min(m-1,d)))
    r = 1 - 1/2**(np.arange(N+1))
    r[-1] = 1
    gamma = 1/2*(r[1:] + r[:-1])
    rho = 3/4*(r[1:] - r[:-1])
    K = np.ceil(3*np.pi*r[1:]/(np.sqrt(5)*rho)).astype(int)
    K[0] = 4
    return gamma, rho, K

# Compute the m-hyperbolic approximation
def hyperbolic_approximation(coeffs, m=30):
    d = coeffs.shape[-1]
    shape = coeffs.shape[:-1]
    gamma, rho, K = disks(d, m)
    N = gamma.size
    Kmax = ((d-1)//K.max()+1)*K.max()
    r = rho/gamma
    D = np.arange(d)
    G = np.zeros(shape + (N, Kmax, m), dtype='complex128')
    P = gamma[:, np.newaxis]**D * coeffs[..., np.newaxis, :]
    G[...,0] = np.fft.fft(P, Kmax)
    for i in range(m-1):
        P *= (D-i)/(i+1) * r[:, np.newaxis]
        G[..., i+1] = np.fft.fft(P[...,i+1:], Kmax)
    return G, gamma, rho, K

# Solve polynomials of small degree
def solve_small(p, m=30, guarantee=True, e=0):
    result = [np.empty(0)]*p.shape[0]
    abs_p = np.abs(p)
    nosol = abs_p[:,0] > abs_p[:,1:].sum(axis=-1)
    unksol = ~nosol
    sols = list(map(np.polynomial.polynomial.polyroots, p[unksol]))
    for i,j in enumerate(np.flatnonzero(unksol)):
        result[j] = sols[i][np.abs(sols[i])<=1]
    if guarantee:
        validate(result, p, e)
    return result

# Guarantee that there is a unique solution nearby
def validate(sols, p, e):
    nonempty = [i for i,x in enumerate(sols) if x.size>0]
    p0 = p[nonempty]
    p1 = np.polynomial.polynomial.polyder(p0, axis=-1)
    p2 = np.polynomial.polynomial.polyder(p1, axis=-1)
    s = np.linalg.norm(p2, 1, axis=-1)
    for i, j in enumerate(nonempty):
       q = 10*s[i]*(np.abs(np.polynomial.polynomial.polyval(sols[j], p0[i]))+e)/\
                   (np.abs(np.polynomial.polynomial.polyval(sols[j], p1[i]))-e)**2
       sols[j] = sols[j][q <= 1]


# Solve using truncated polynomials
def solve_piecewise(G, gamma, rho, K, m=30, rtol=8, guarantee=True, e=0):
    result = np.array([],dtype='complex128')
    Kmax = G.shape[1]
    for p, g, r, Kn in zip(G,gamma,rho,K):
        step = (Kmax-1)//(Kn-1)  # step * (Kn-1) < Kmax
        w = np.exp(-2j*np.pi*np.arange(0,Kmax,step)/Kmax)
        sols = solve_small(p[::step], m, guarantee, e)
        for i in range((Kmax-1)//step + 1):
            sols[i] = g*w[i] + r*sols[i]
            result = np.append(result, sols[i])
    rounded = np.round(result, decimals=-int(np.log10(rtol)))
    _, ind = np.unique(rounded, return_index=True)
    return result[ind]

# truncate and solve a polynomial over the complex
def solve(p, m=30, rtol=None, guarantee=True):
    rtol =  max(3*2**(-m), 2**-35) if rtol is None else rtol
    dtype = p.dtype if hasattr(p, 'dtype') else 'complex128'
    p = np.trim_zeros(p, 'b')
    coeffs = np.zeros((2, len(p)), dtype=dtype)
    coeffs[0] = p
    coeffs[1] = coeffs[0,::-1]
    G, gamma, rho, K = hyperbolic_approximation(coeffs, m)
    e = 3*np.linalg.norm(coeffs[0], 1)*(m+2)/2**m
    sols = solve_piecewise(G[0], gamma, rho, K, m, rtol, guarantee, e)
    invsols = solve_piecewise(G[1], gamma, rho, K, m, rtol, guarantee, e)
    result = np.concatenate([sols , 1/invsols])
    rounded = np.round(result, decimals=-int(np.log10(rtol)))
    _, ind = np.unique(rounded, return_index=True)
    return result[ind]
```

Roots distribution

«*Polynomial rootfinding is an ill-conditioned problem in general*»
Trefethen and Bau

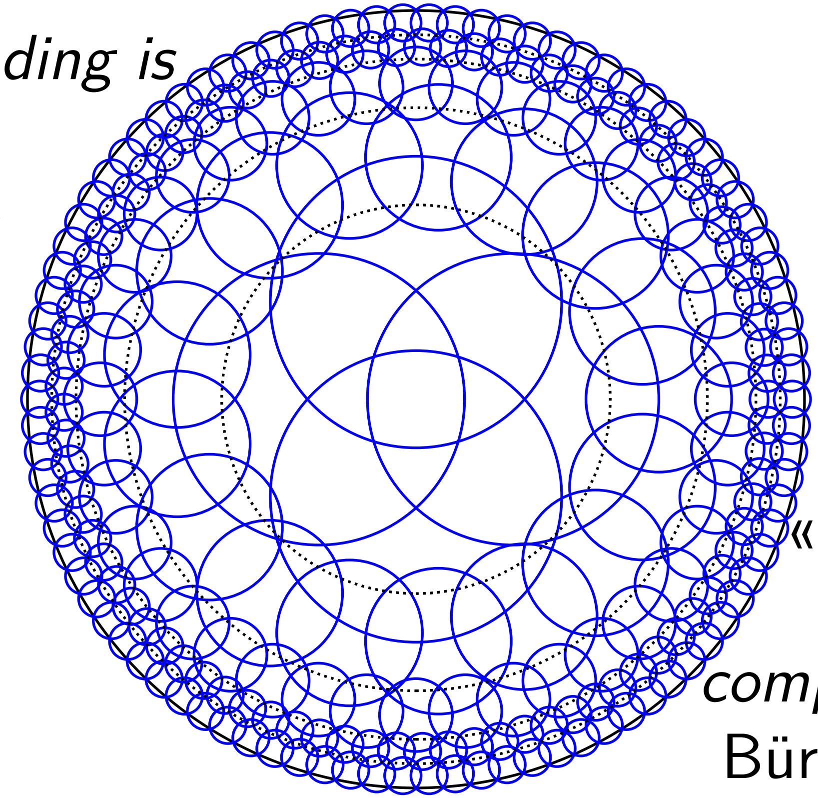


«*Typical polynomials are well-conditioned for the computation of their zeros*»
Bürgisser, Cucker, Cardozo

Roots distribution

«*Polynomial rootfinding is an ill-conditioned problem in general*»
Trefethen and Bau

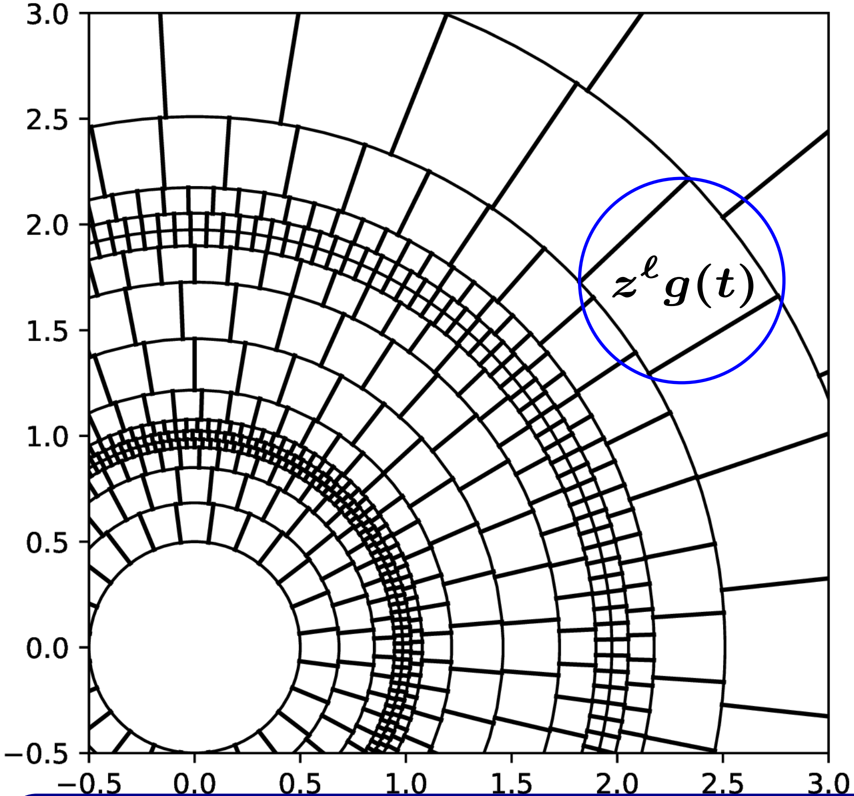
For uniform distribution of roots



For uniform distribution of coefficients

«*Typical polynomials are well-conditioned for the computation of their zeros*»
Bürgisser, Cucker, Cardozo

Relative approximation



$$f(z) = f_0 + \cdots + f_d z^d$$

$$\widetilde{f}(z) = |f_0| + \cdots + |f_d| z^d$$

Sector enclosed in the disk of center γ and radius ρ

$$z := \gamma + \rho t$$

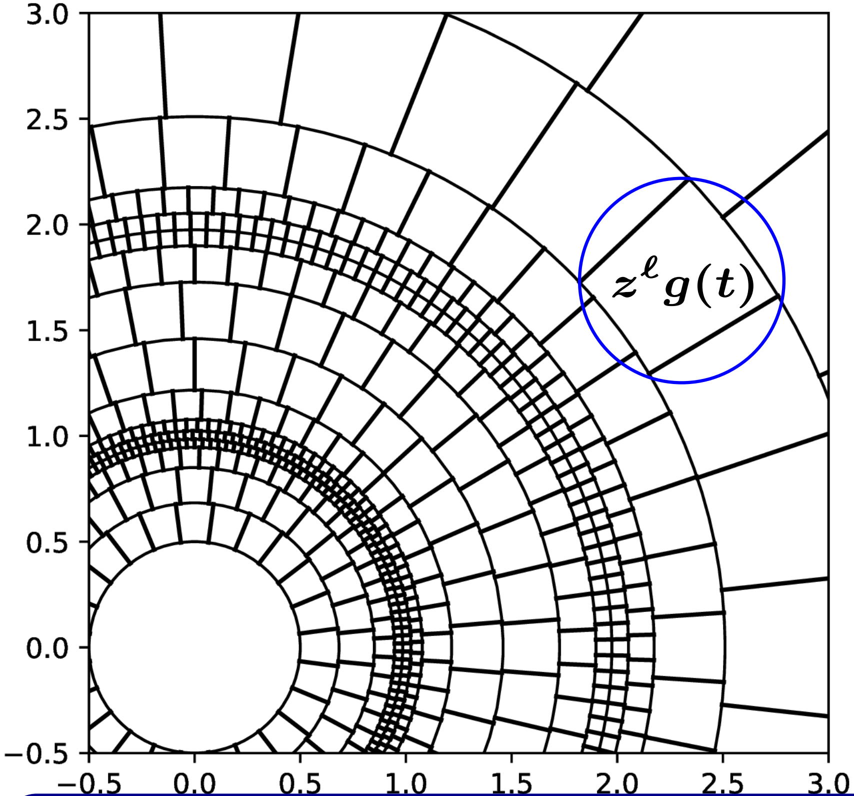
Theorem (relative approximation) [Imbach,M. 2023]

It is possible to compute all g of degree m satisfying

$$|f(z) - z^\ell g(t)| < 2^{-m} \widetilde{f}(|z|)$$

in $\widetilde{O}(d(m + \log \tau))$ bit operations

Corollary



$$f(z) = f_0 + \cdots + f_d z^d$$

$$\widetilde{f}(z) = |f_0| + \cdots + |f_d| z^d$$

Sector enclosed in the disk of center γ and radius ρ

$$z := \gamma + \rho t$$

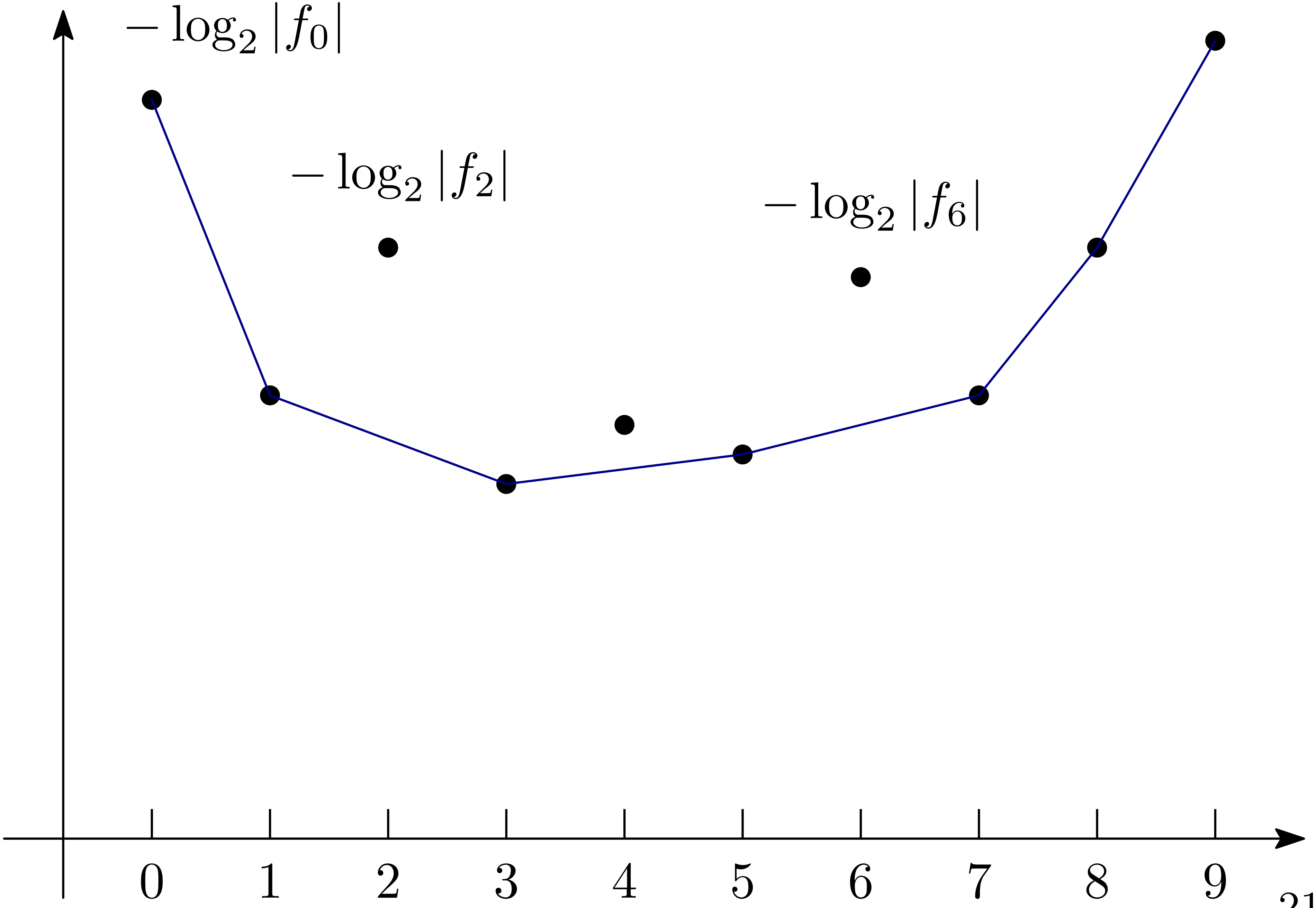
Fast evaluation

With error $2^{-m}\widetilde{f}(|z|)$
in
$\widetilde{O}(m(m + \log \tau))$

Fast root finding

All roots with m bits
in
$\widetilde{O}(d(m + \log \tau + \log \kappa))$

Newton polygon



Newton polygon

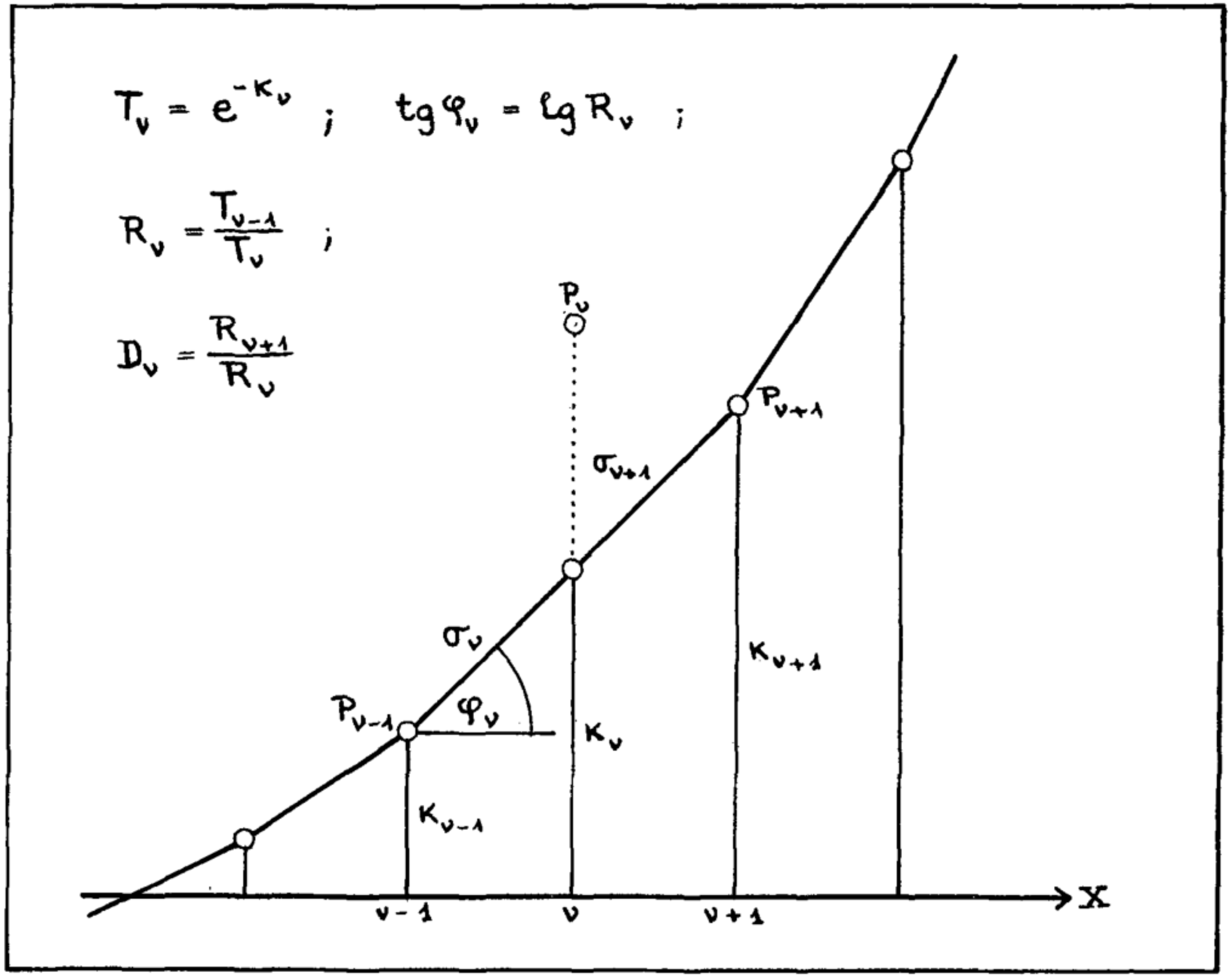


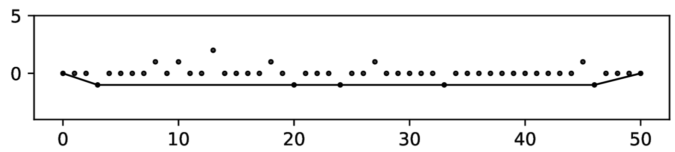
Fig. 1.

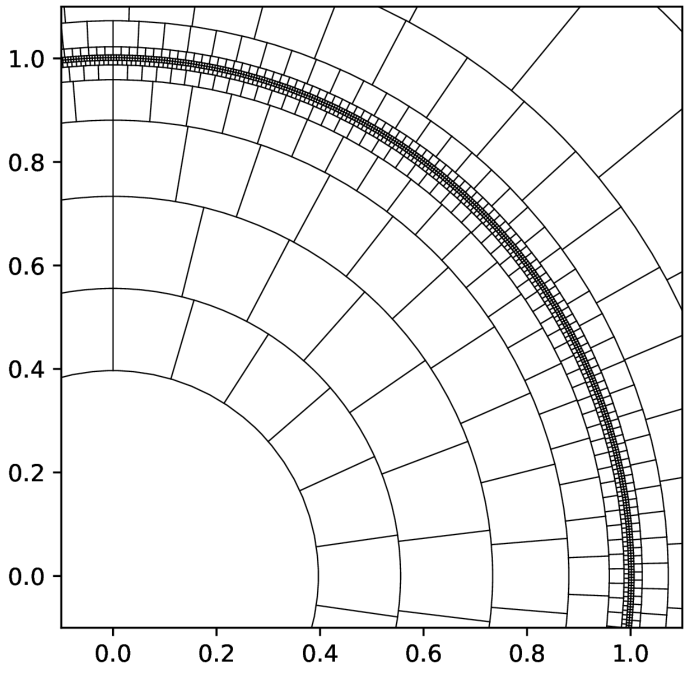
Alexandre Ostrowski. *Recherches sur la méthode de graeffe et les zéros des polynomes et des séries de laurent.* Acta Mathematica, 1940

Examples

Hyperbolic

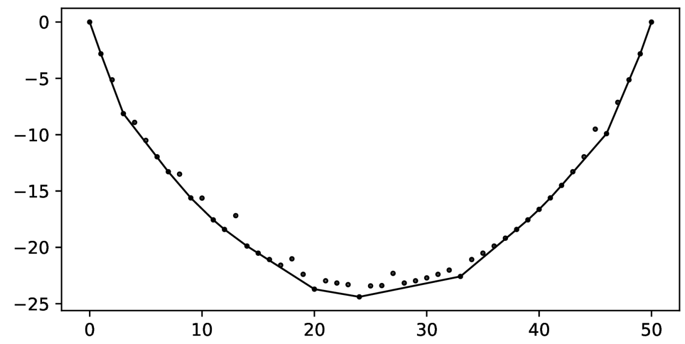
$$\sum c_j z^j$$

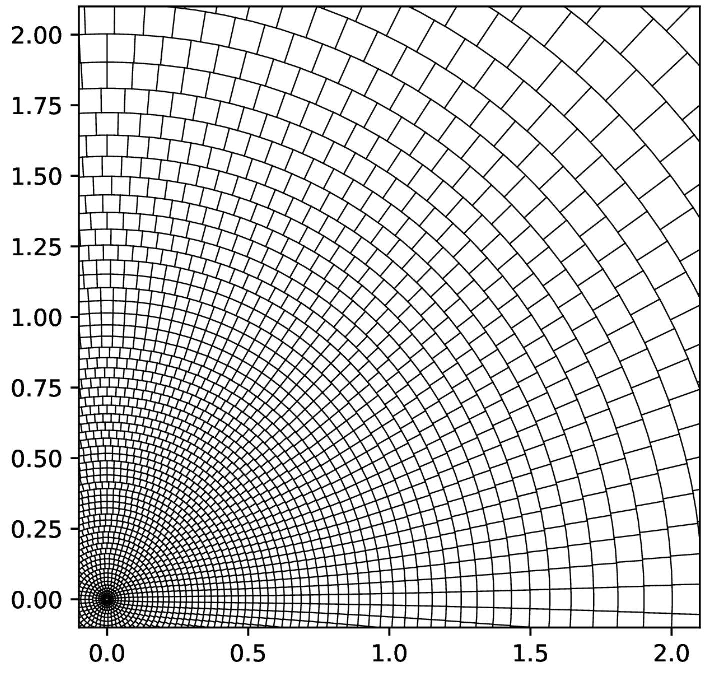




Elliptic

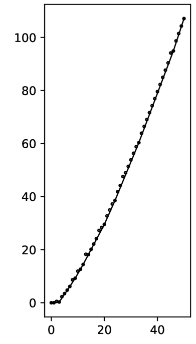
$$\sum c_j \sqrt{\binom{d}{j}} z^j$$

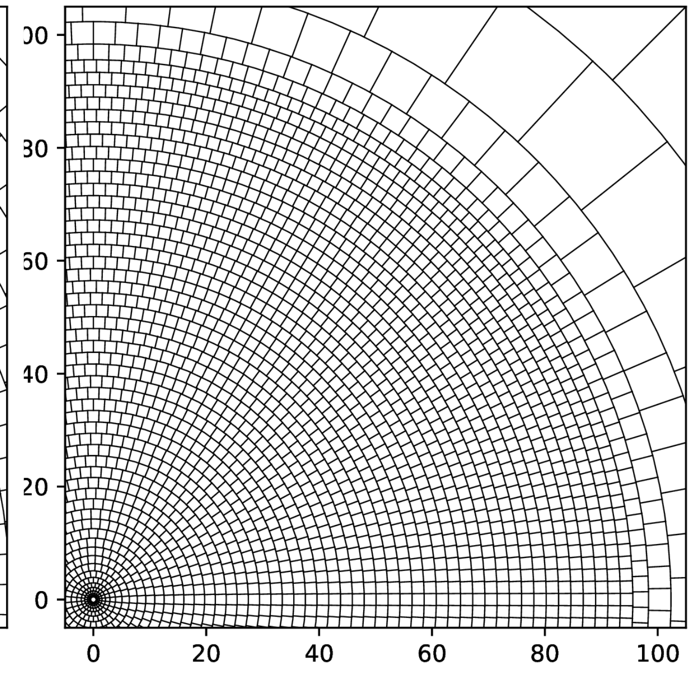




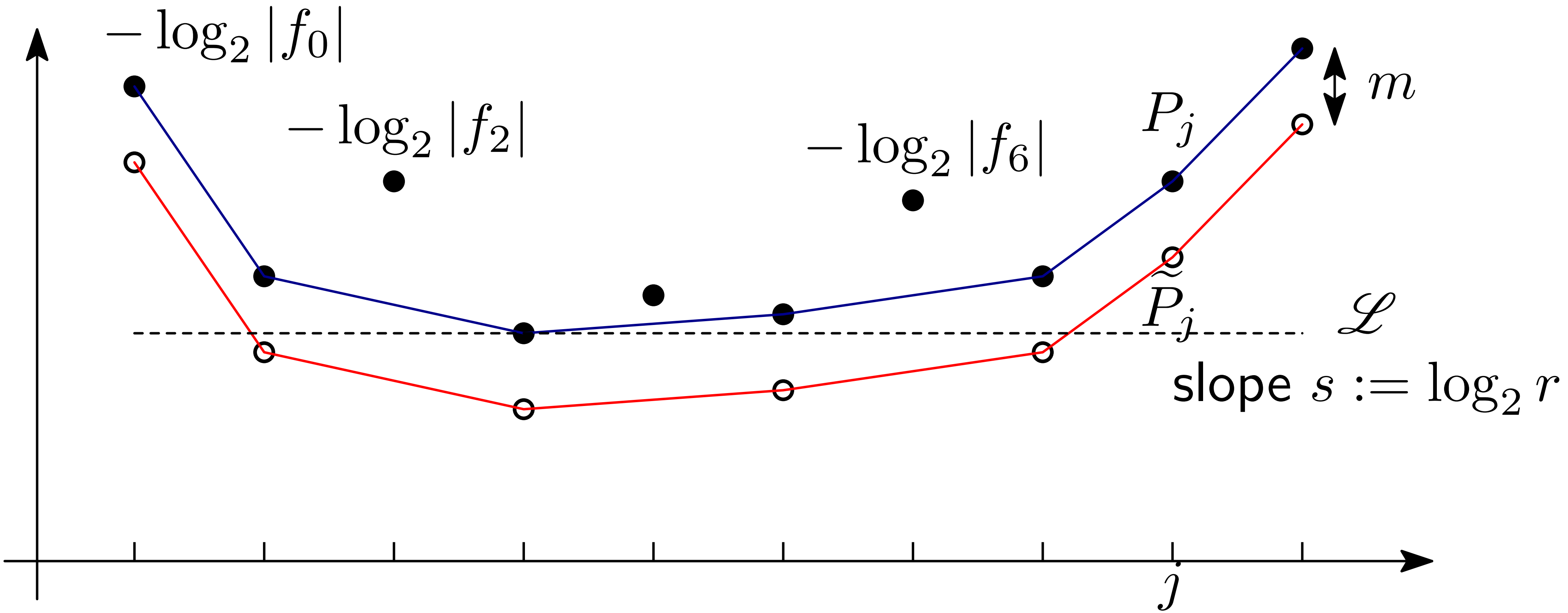
Flat

$$\sum c_j \sqrt{\frac{1}{j!}} z^j$$





Newton polygon

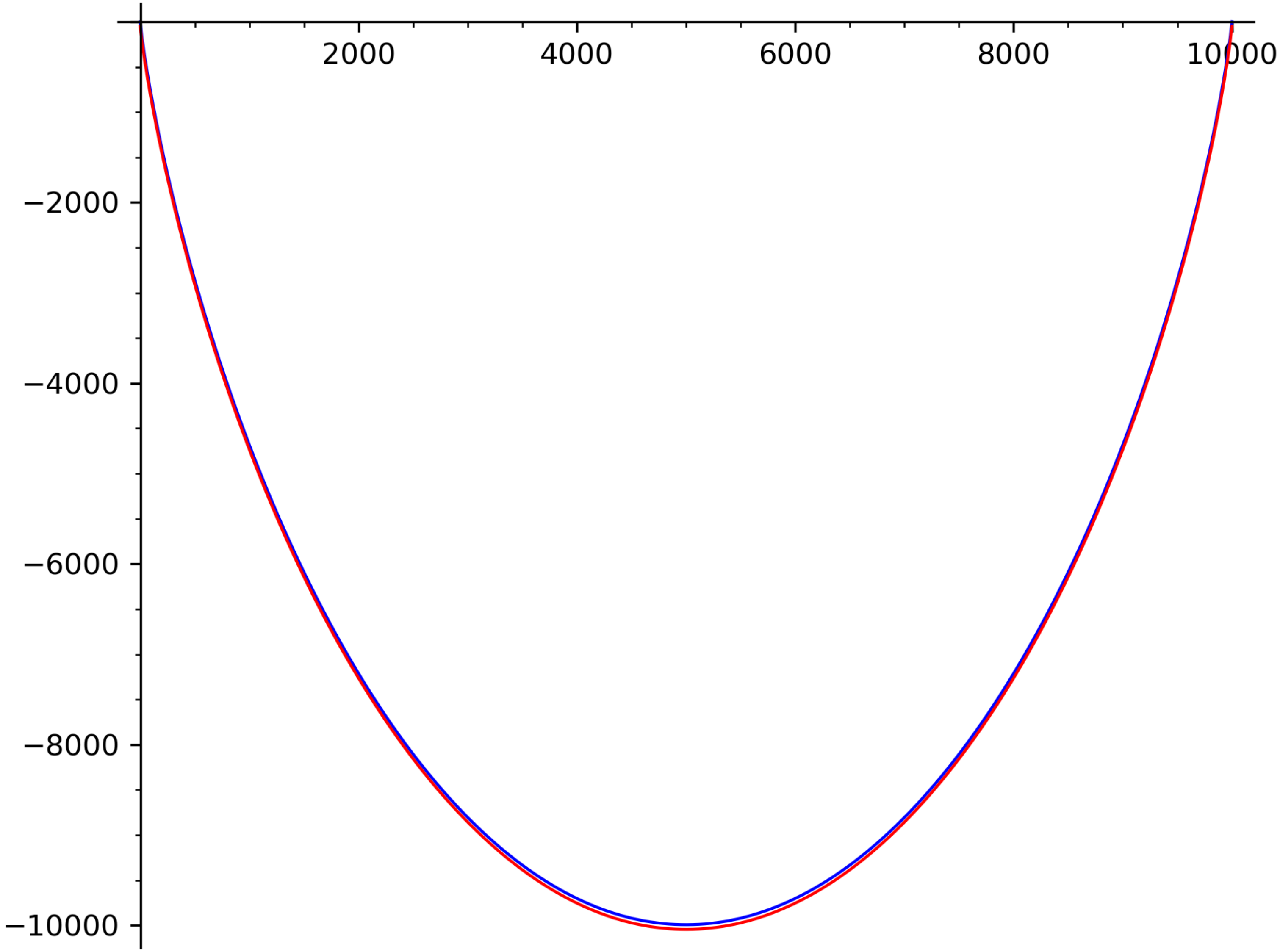


Fast evaluation

$\widetilde{P_j}$ above $\mathscr{L}$ $\iff$ $|f_j|r^j < 2^{-m}\widetilde{f}(r)$

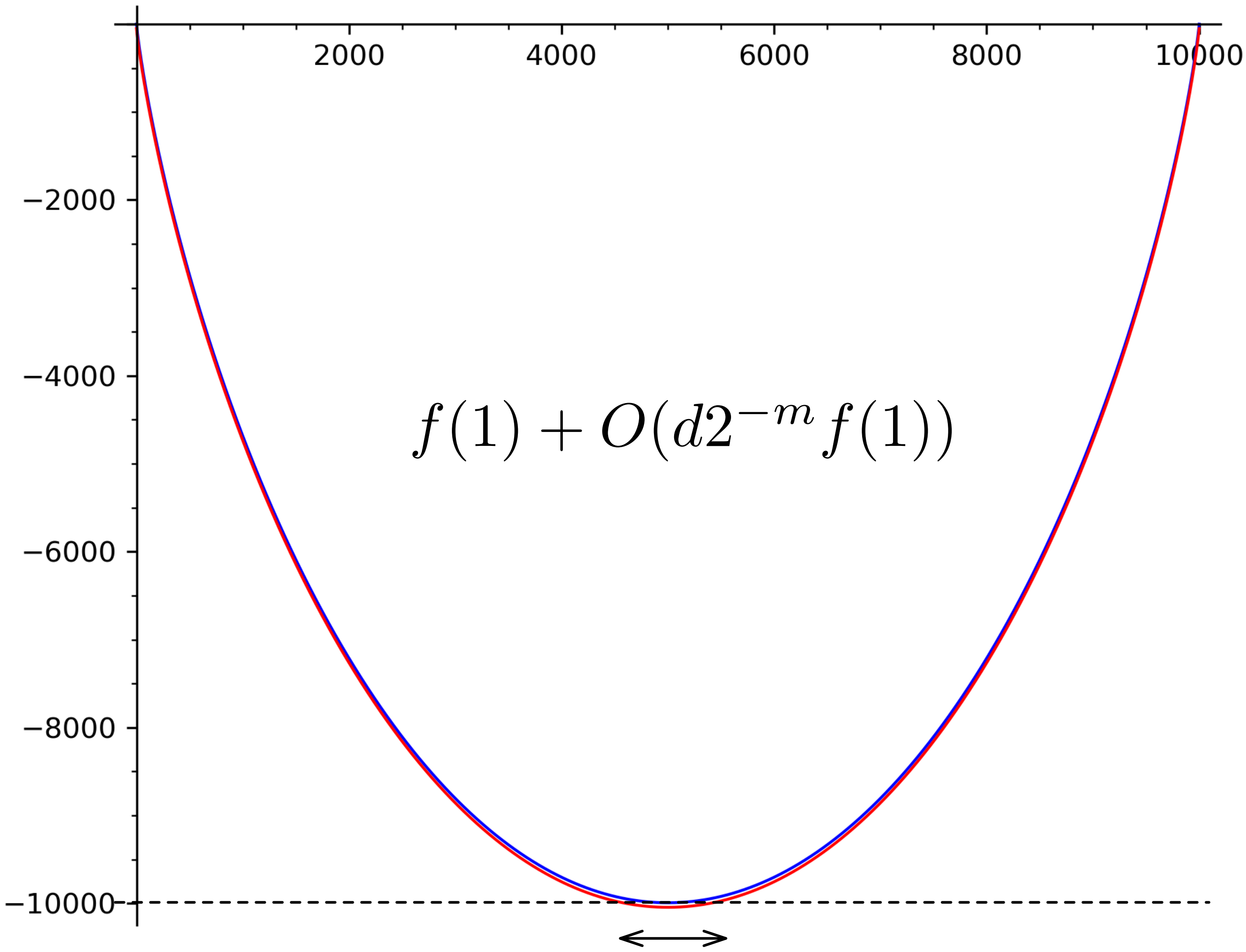
Example: selecting monomials

$$f(z) = (1+z)^{10000} = \sum \binom{10000}{j} z^j$$



Example: selecting monomials

$$f(z) = (1+z)^{10000} = \sum \binom{10000}{j} z^j$$



$k = 864$ monomials

Example: Taylor expansion

$$1$$

$$10\,000z$$

$$\vdots$$

$$\binom{10\,000}{\ell} z^{\ell}$$

$$\binom{10\,000}{j} z^{j}$$

$$\binom{10\,000}{u} z^{u}$$

$$\vdots$$

$$z^{10\,000}$$

Example: Taylor expansion

$$1 \qquad (1+\varepsilon t)^k = 1 + \cdots + \binom{k}{m}\varepsilon^m t^m + \mathcal{O}\left(\frac{k}{m}\varepsilon\right)^m$$

$$10\,000z$$

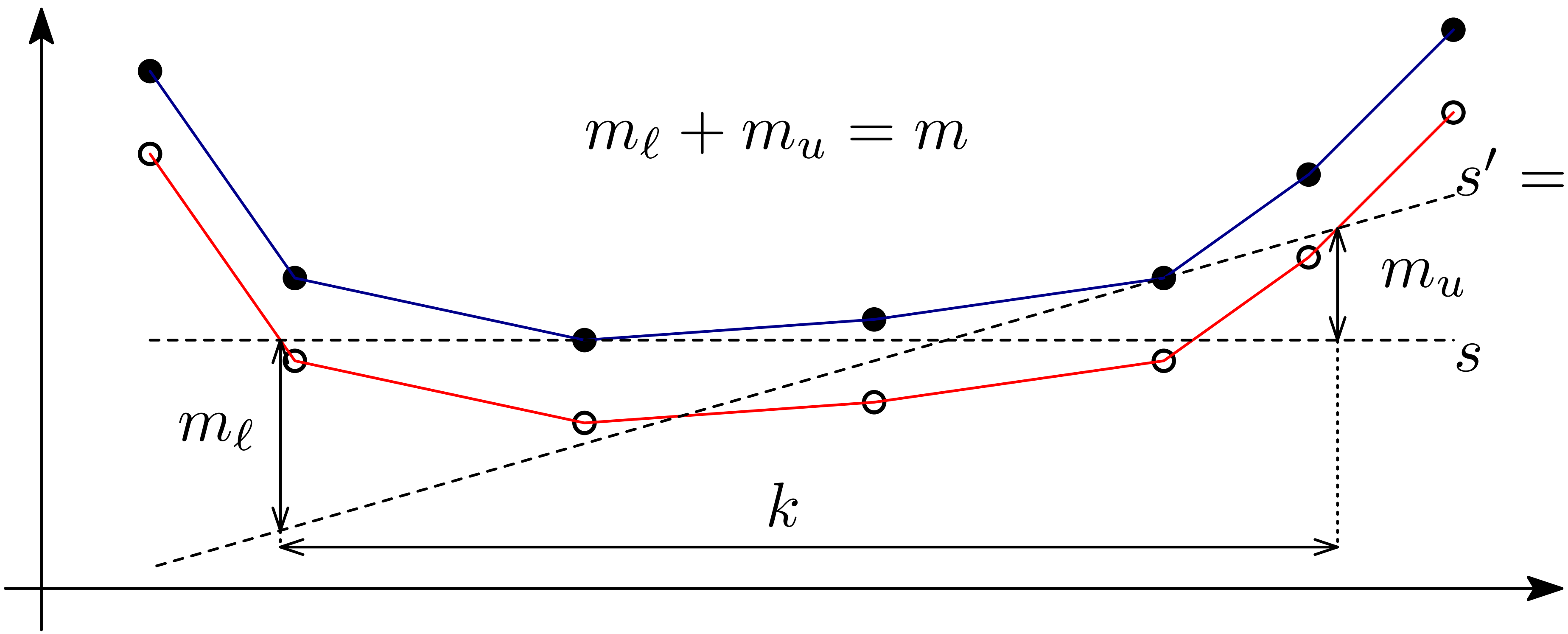
$$\vdots$$

$$k\left\{\begin{aligned}\binom{10\,000}{\ell} z^\ell &= \binom{10\,000}{\ell} z^\ell \\ \binom{10\,000}{j} z^j &= \binom{10\,000}{j} z^\ell (1+\varepsilon t)^{j-\ell} \\ \binom{10\,000}{u} z^u &= \binom{10\,000}{u} z^\ell (1+\varepsilon t)^{u-\ell}\end{aligned}\right.$$

$$\vdots$$

$$z^{10\,000}$$

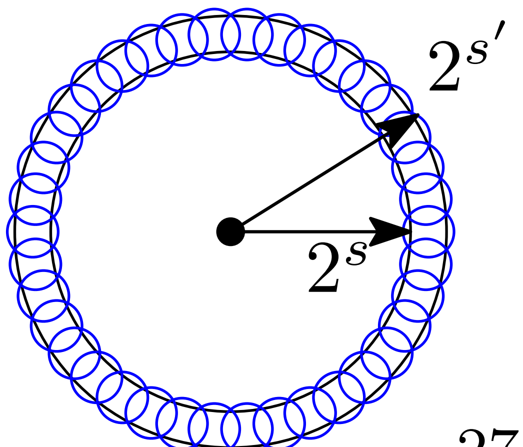
Algorithm: one ring



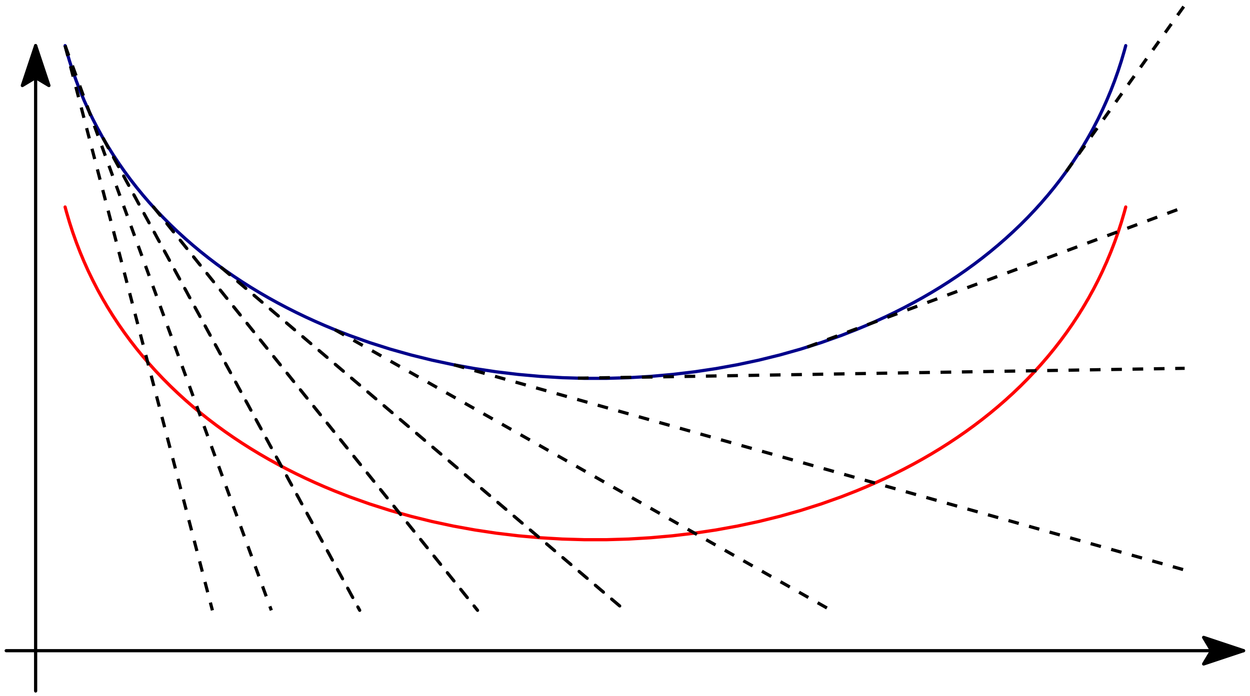
Polynomial approximations

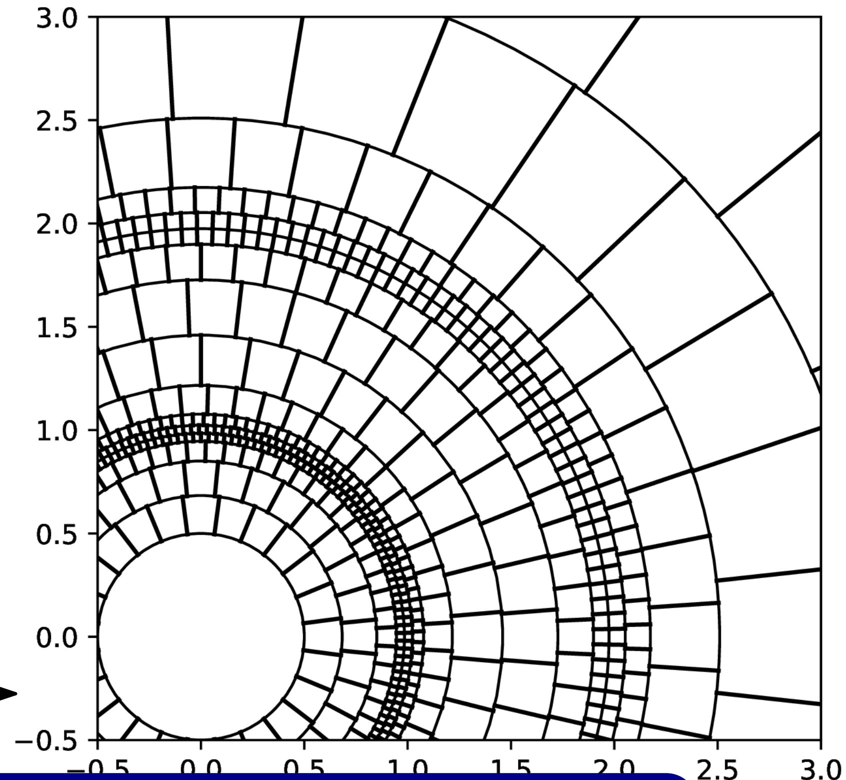
- $O(k/m)$ approximations $z^\ell g(t)$
- $\deg g$ in $O(m)$
- Computed in $\widetilde{O}(km)$

Ring $R(2^s, 2^{s'})$



Algorithm: all rings





Approximation algorithm

WHILE $s_j < s_{max}$

- $s_{j+1} = s_j + \frac{m}{k_j}$
- Compute $O(\frac{k}{m})$ approximations on $R(2^{s_j}, 2^{s_{j+1}})$

Complexity

Theorem

The piecewise approximation algorithm costs

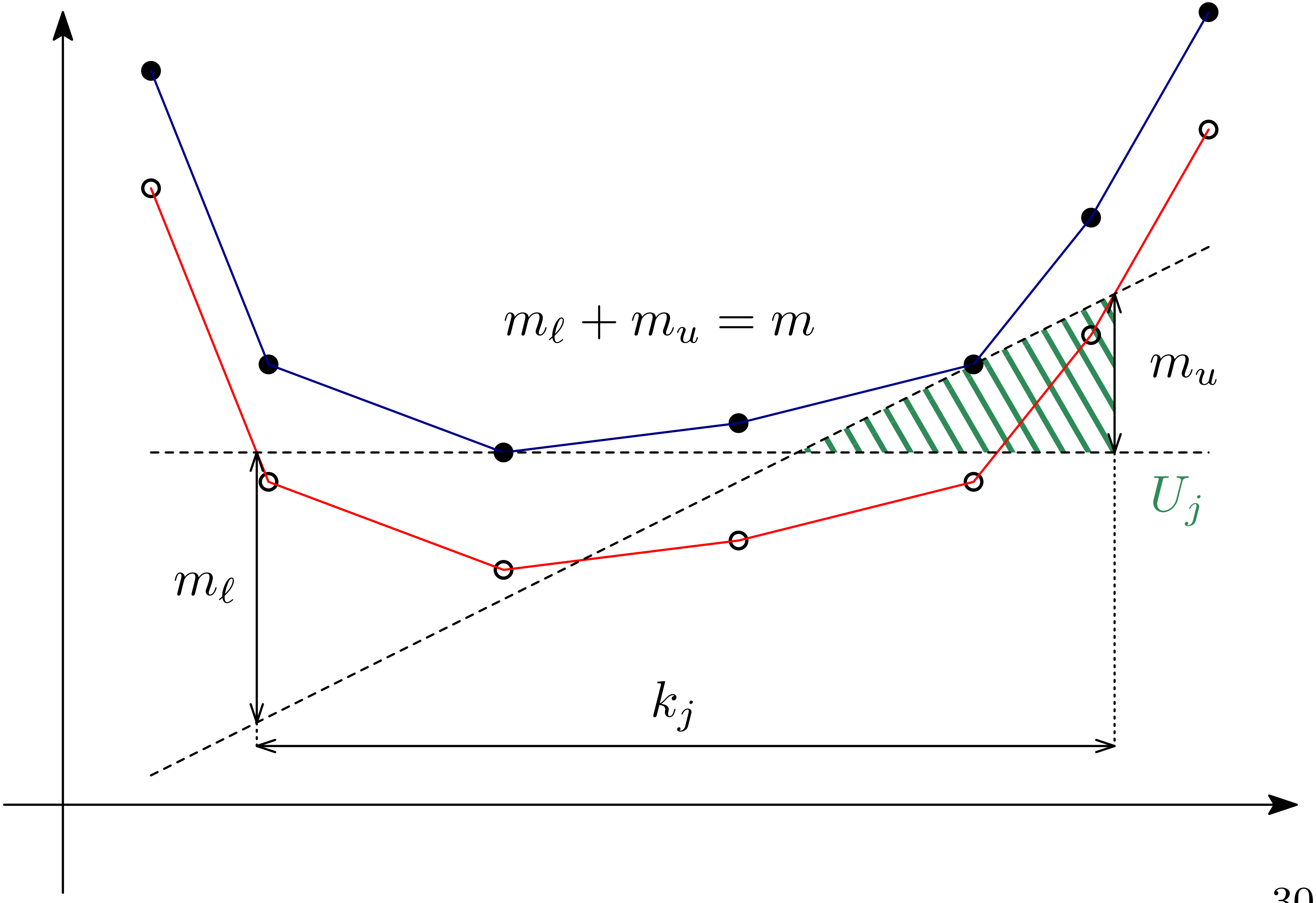
$$\tilde{\mathcal{O}}\left(\sum k_j m\right) \in \tilde{\mathcal{O}}(dm) \text{ bit-operations}$$

Approximation algorithm

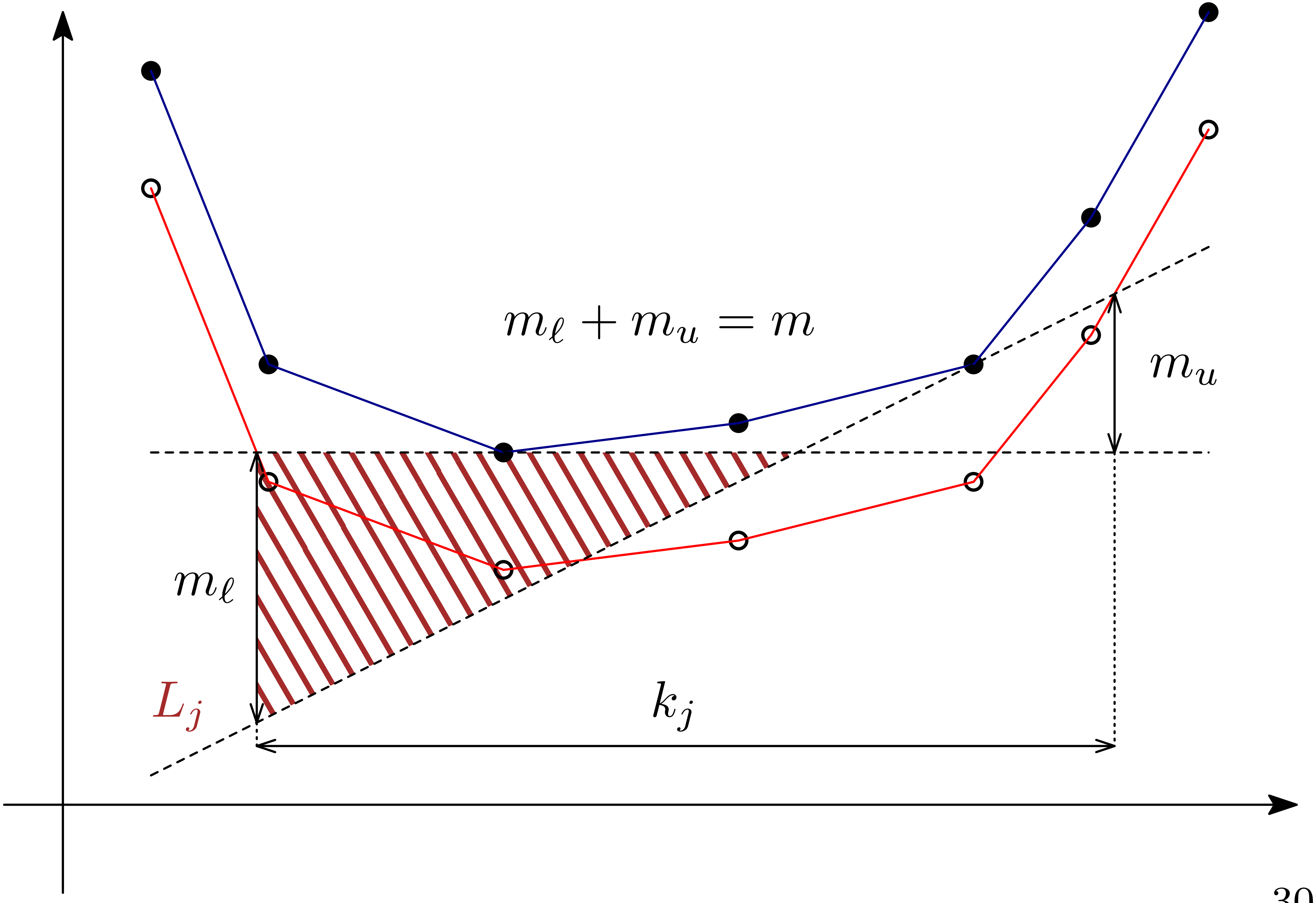
$\textsc{While } s_j < s_{max}$

- $s_{j+1} = s_j + \frac{m}{k_j}$
- Compute $O(\frac{k}{m})$ approximations on $R(2^{s_j}, 2^{s_{j+1}})$

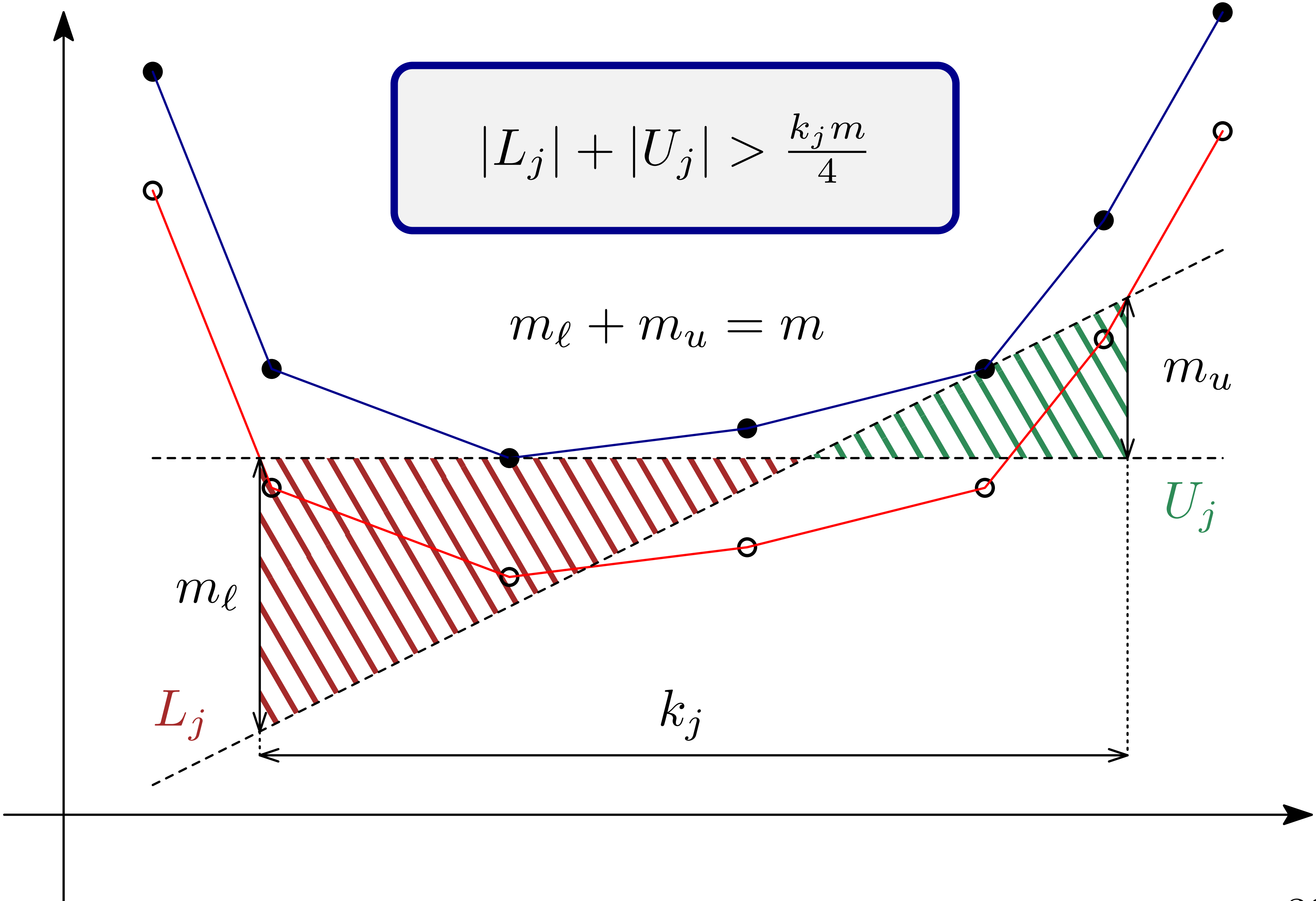
Proof



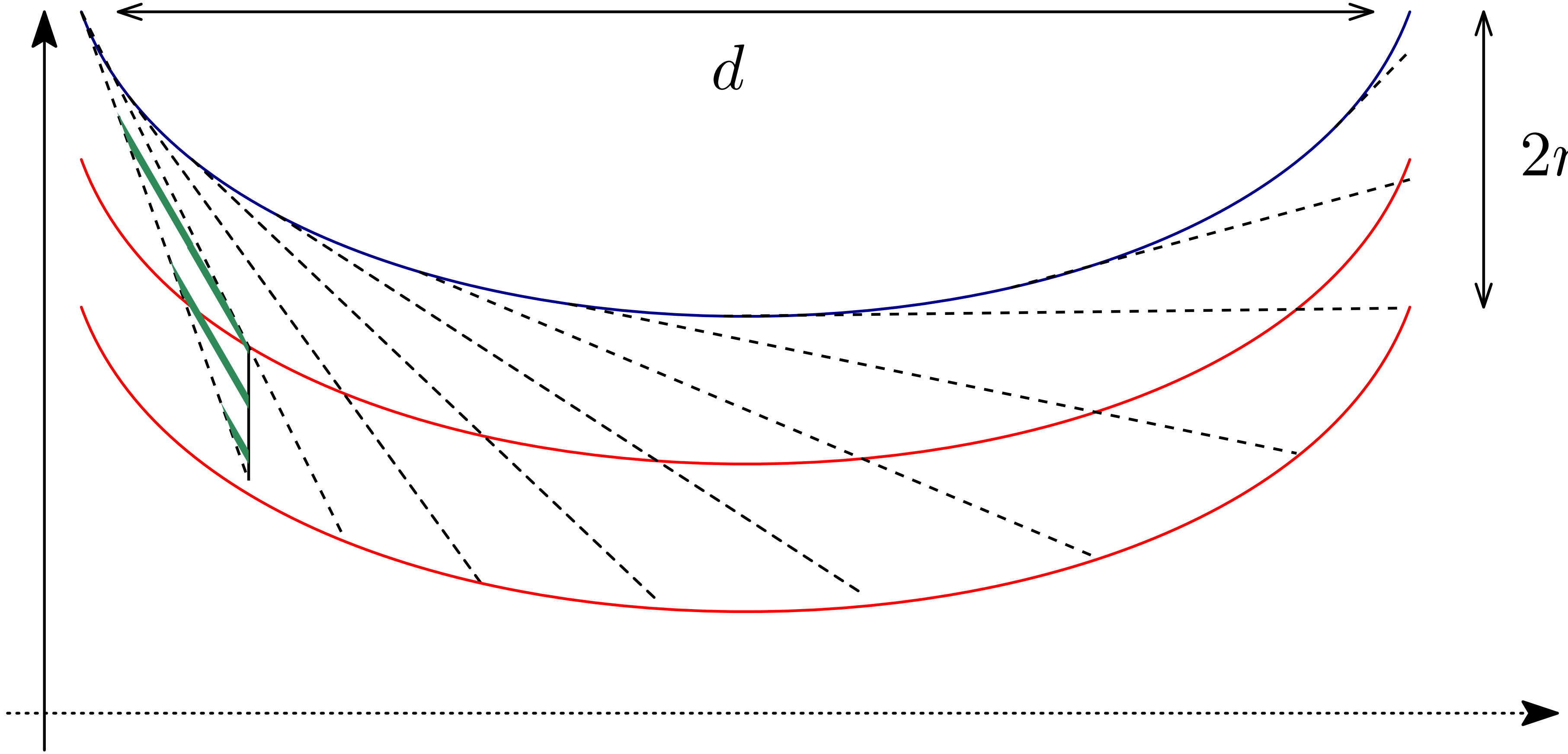
Proof



Proof



Proof

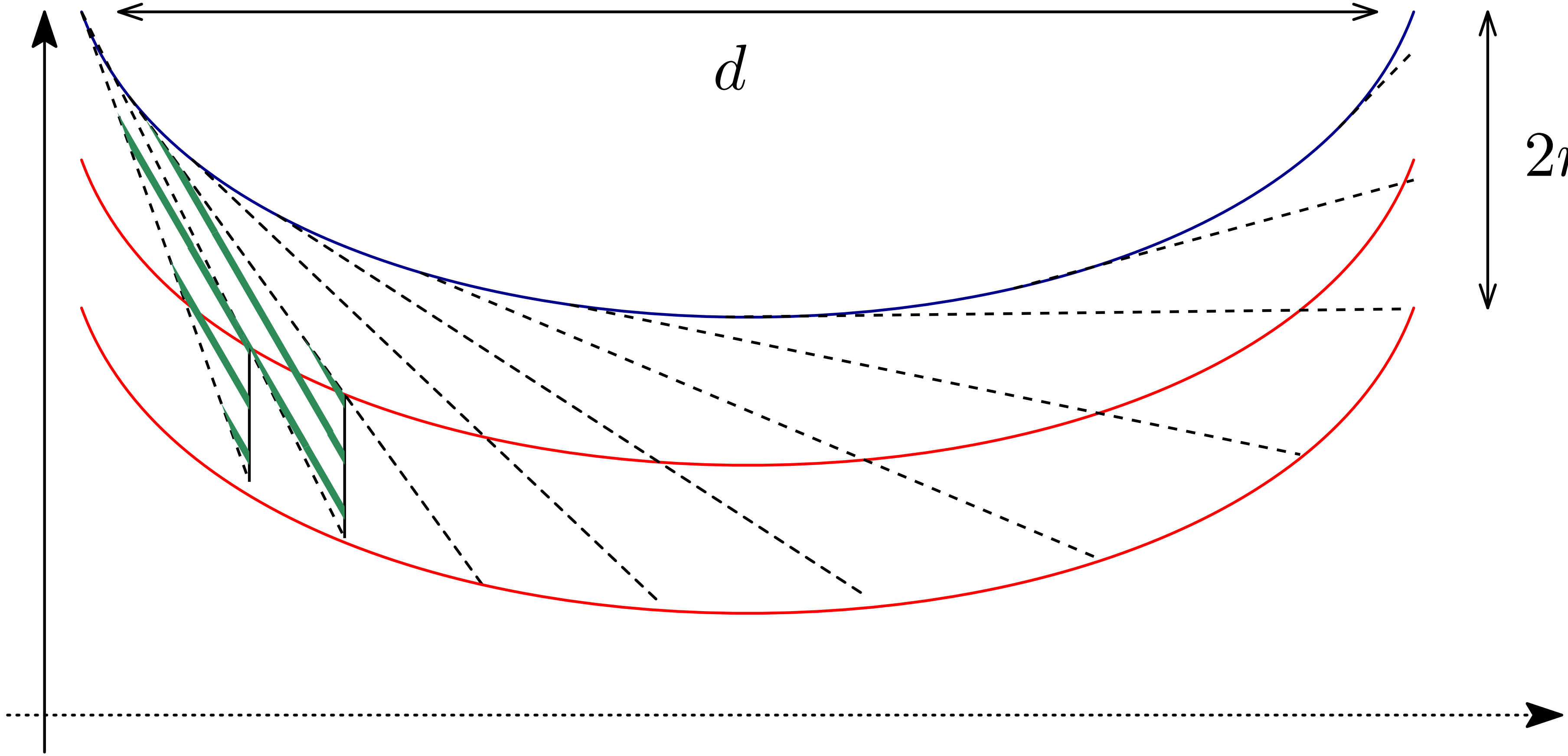


U_j piecewise disjoint

Union of U_j in B_{2m}

$$\boxed{\sum |U_j| < 2dm}$$

Proof

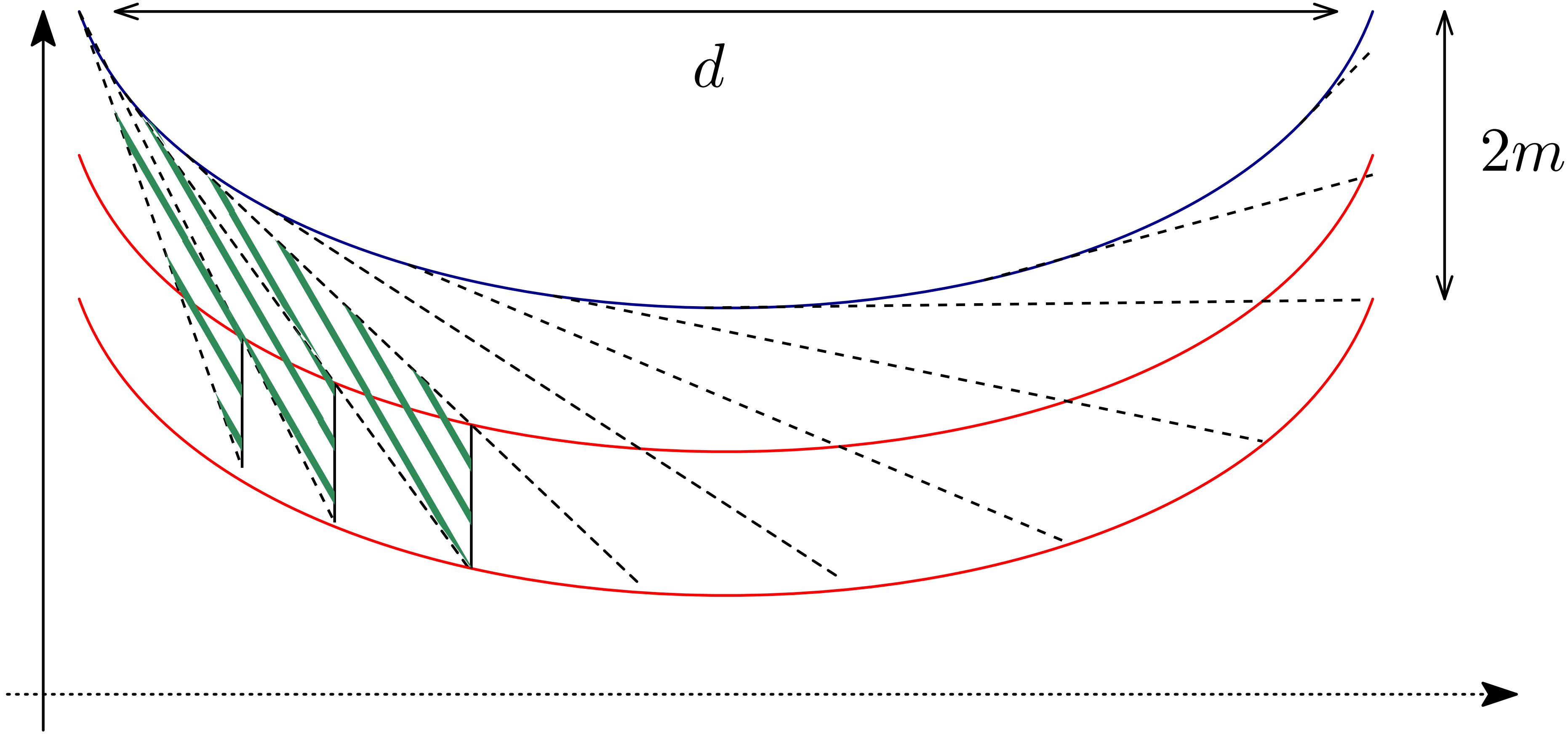


U_j piecewise disjoint

Union of U_j in B_{2m}

$$\boxed{\sum |U_j| < 2dm}$$

Proof

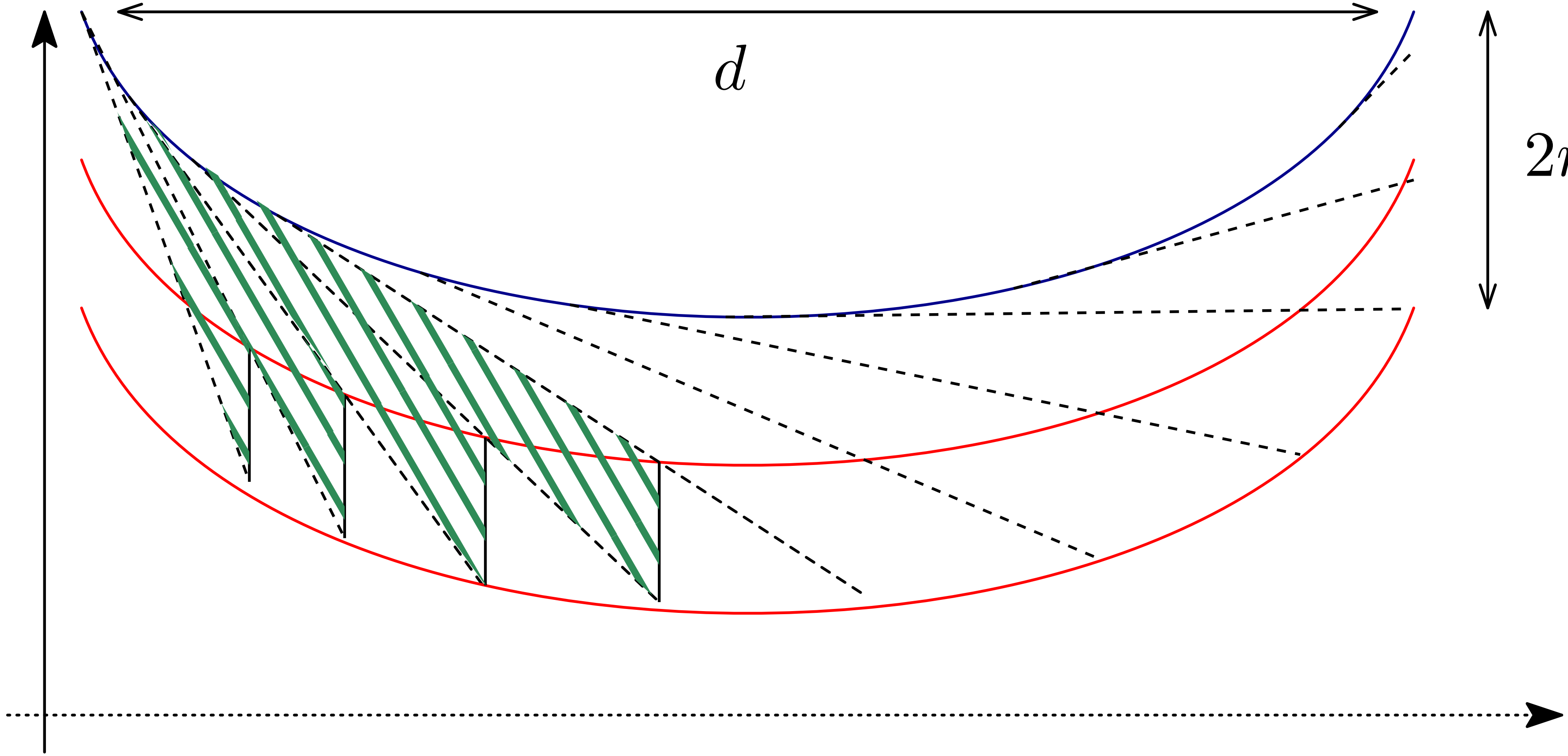


U_j piecewise disjoint

Union of U_j in B_{2m}

$$\sum |U_j| < 2dm$$

Proof

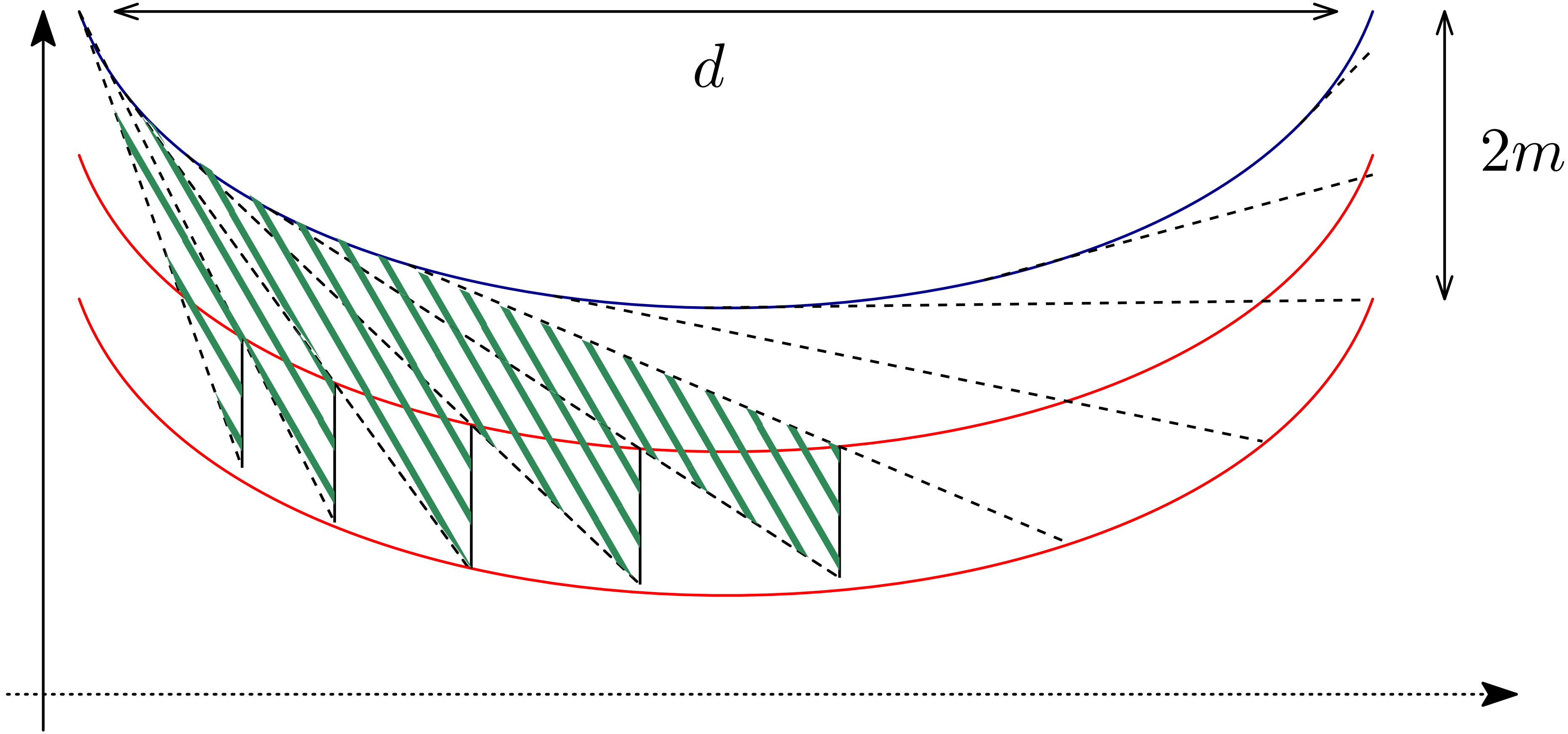


U_j piecewise disjoint

Union of U_j in B_{2m}

$$\sum |U_j| < 2dm$$

Proof

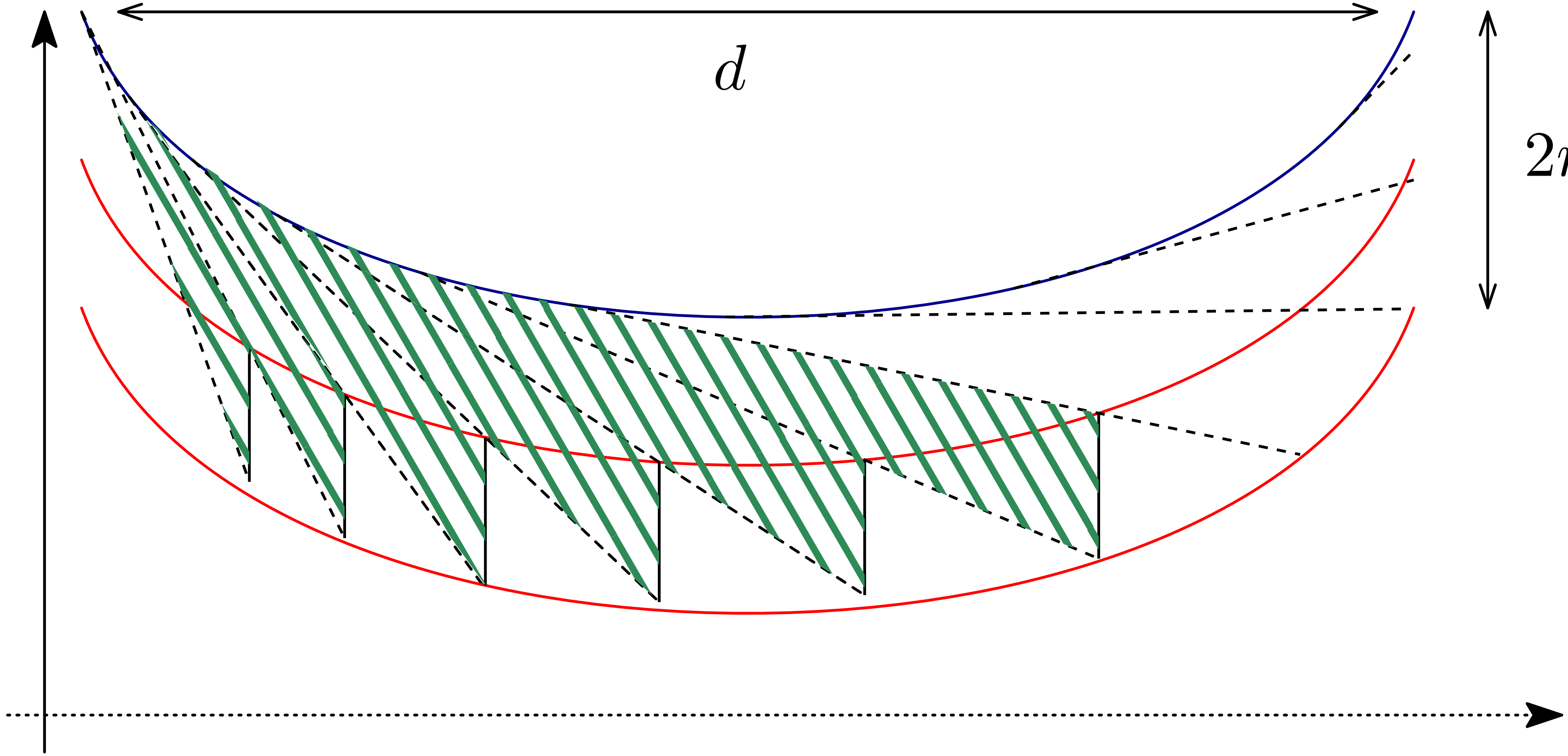


U_j piecewise disjoint

Union of U_j in B_{2m}

$$\sum |U_j| < 2dm$$

Proof

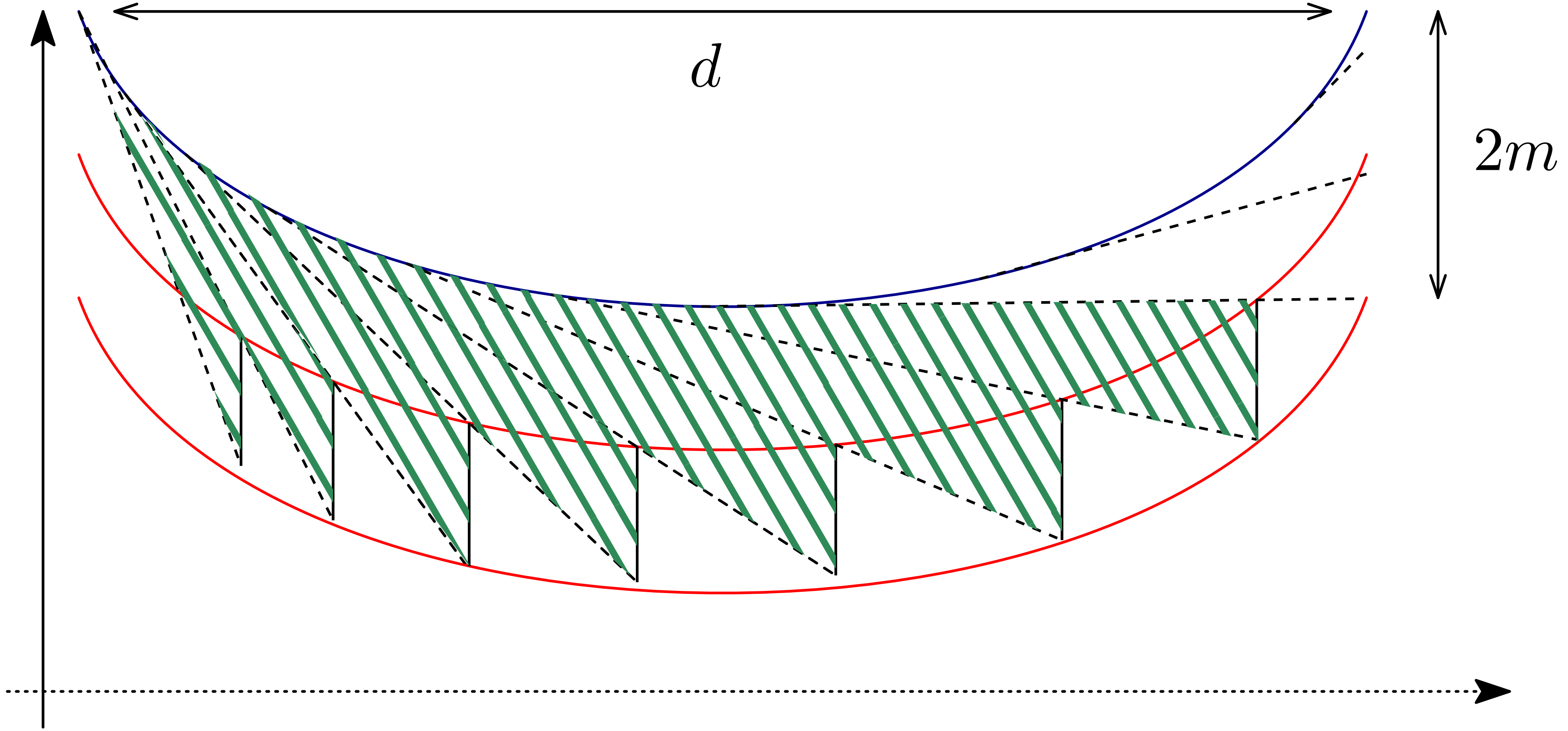


U_j piecewise disjoint

Union of U_j in B_{2m}

$$\sum |U_j| < 2dm$$

Proof

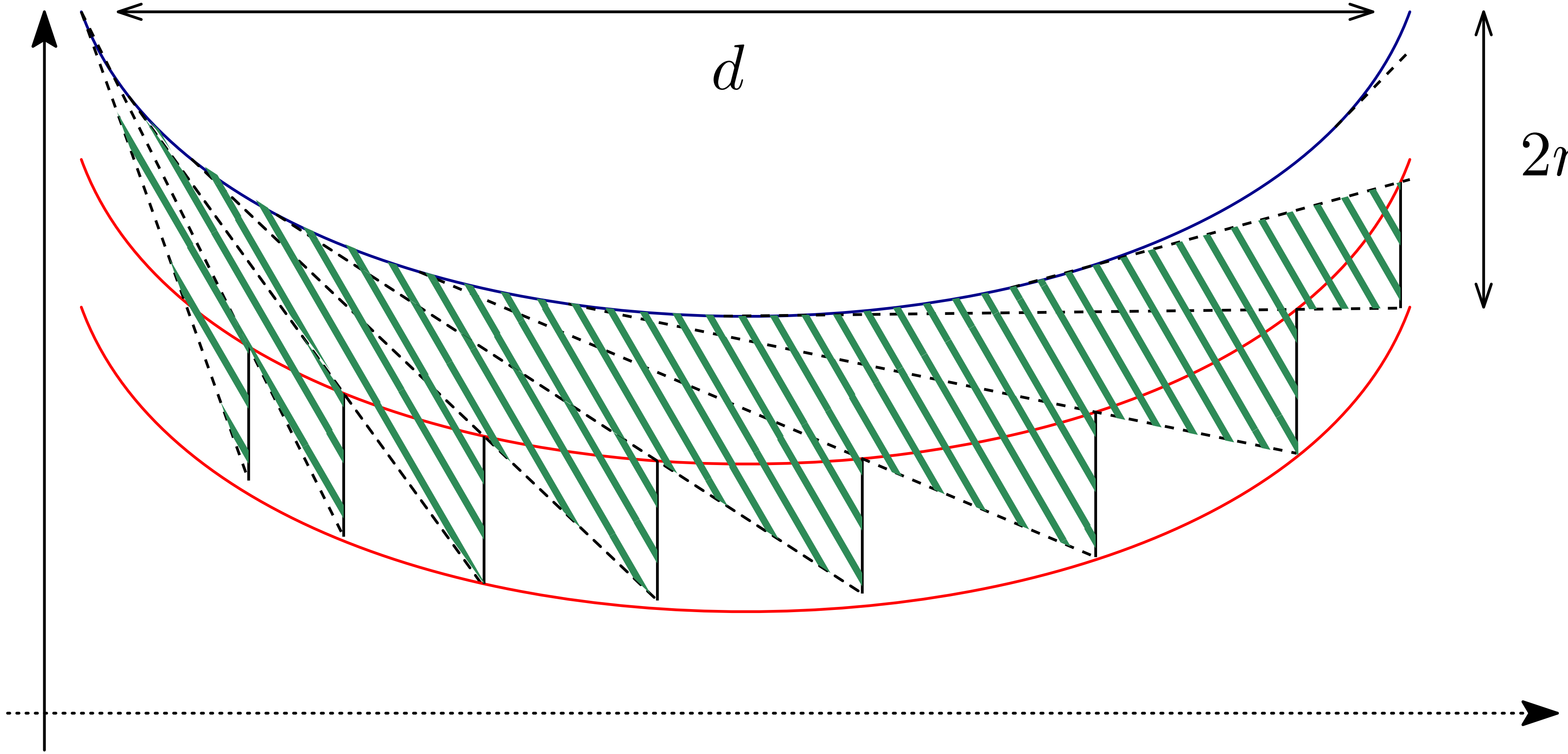


U_j piecewise disjoint

Union of U_j in B_{2m}

$$\boxed{\sum |U_j| < 2dm}$$

Proof

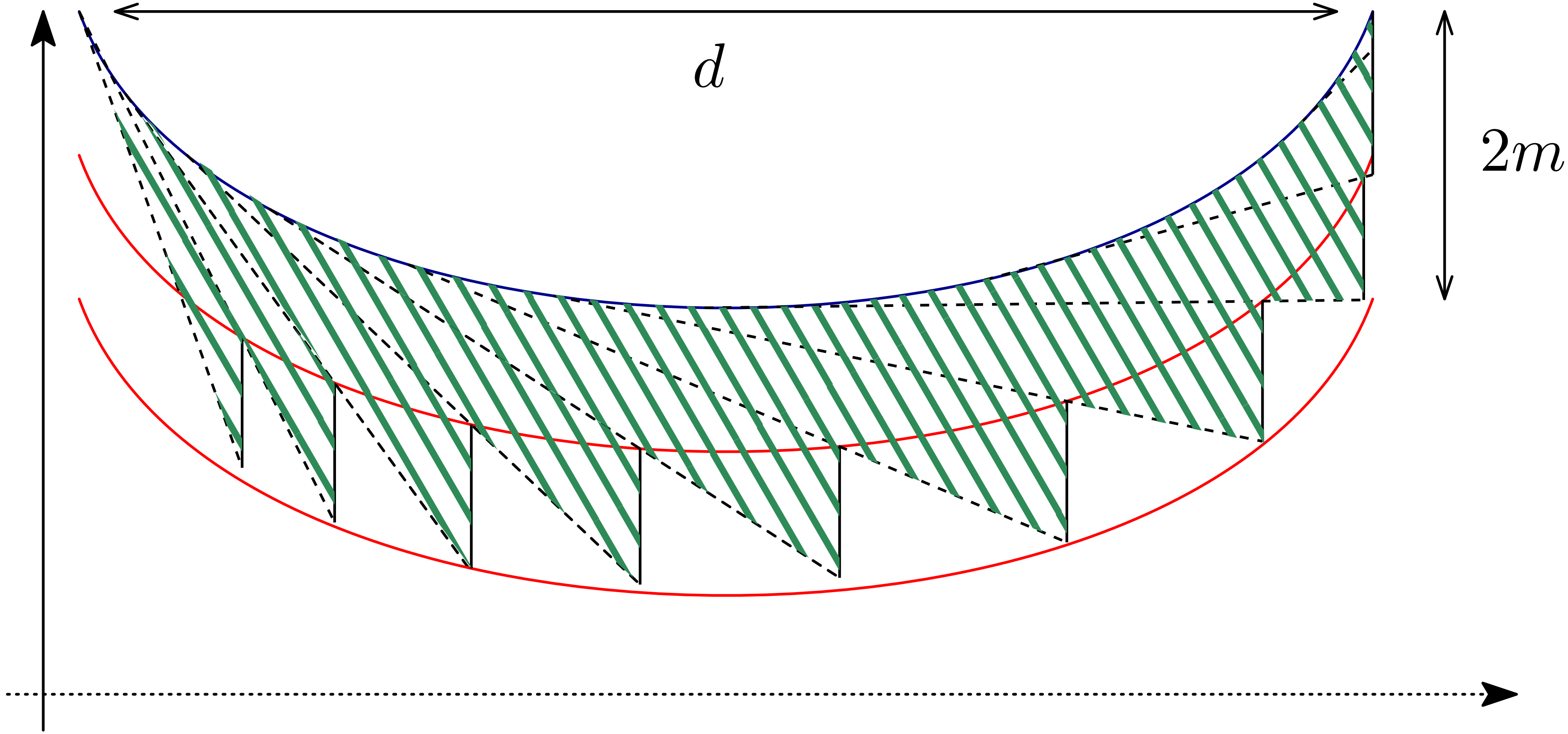


U_j piecewise disjoint

Union of U_j in B_{2m}

$$\boxed{\sum |U_j| < 2dm}$$

Proof



U_j piecewise disjoint

Union of U_j in B_{2m}

$$\sum |U_j| < 2dm$$

Proof

$$|L_j| + |U_j| > \frac{k_j m}{4}$$

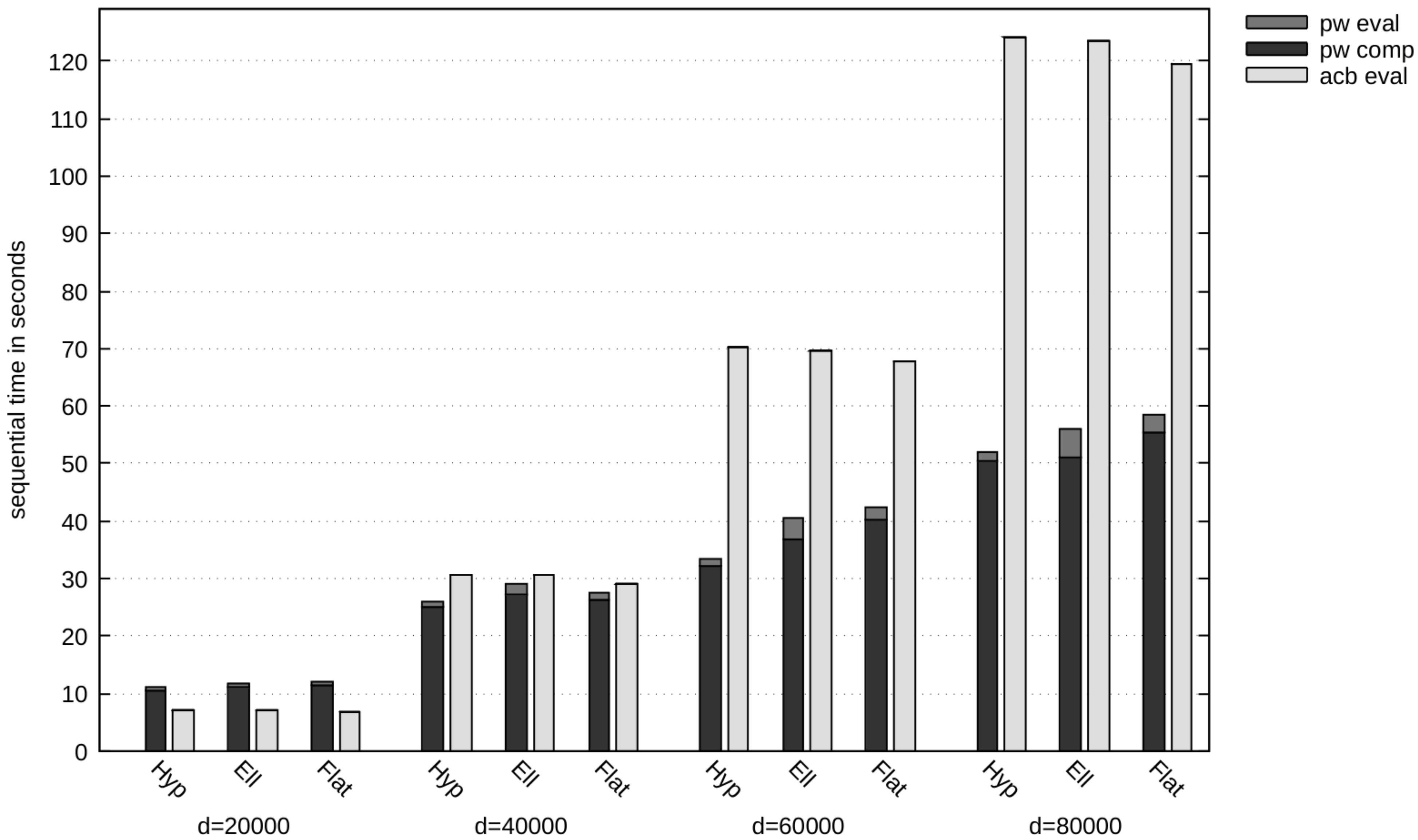
$$\sum |L_j| < 2dm$$

$$\sum |U_j| < 2dm$$

Main lemma

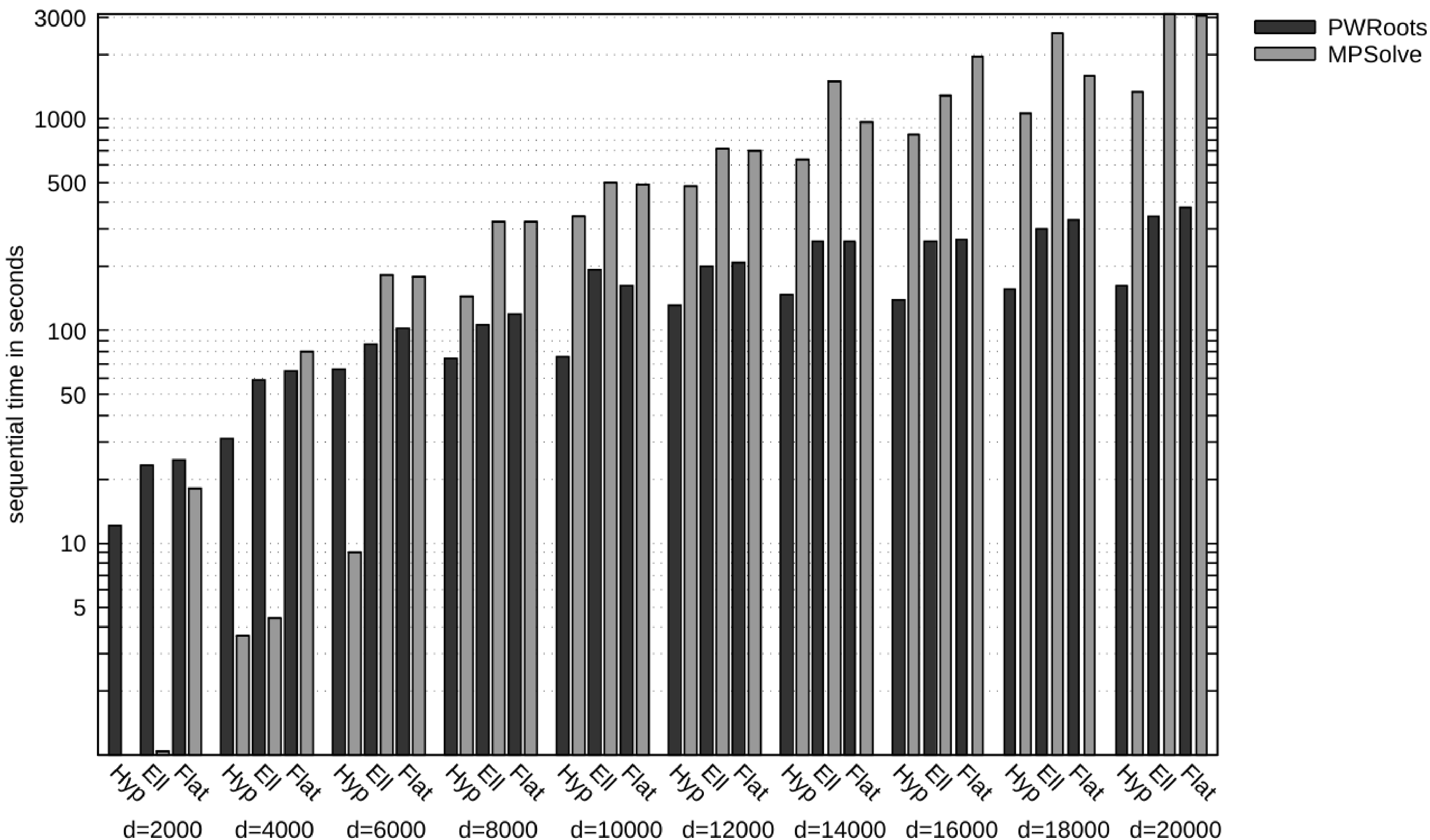
$$\sum \frac{k_j m}{4} \leq \sum |L_j| + \sum |U_j| \leq 4dm$$

Benchmark



pw eval
pw comp
acb eval
sequential time in seconds
0
10
20
30
40
50
60
70
80
90
100
110
120
Hyp
Ell
Flat
d=20000
d=40000
d=60000
d=80000

Benchmark



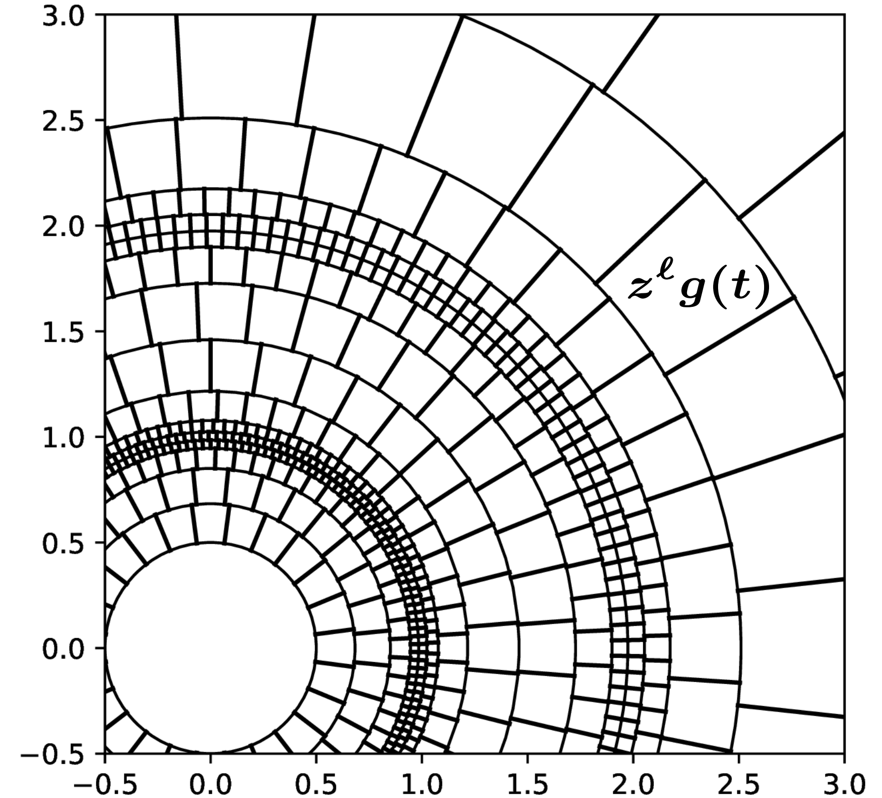
3000
1000
500
100
50
10
5
sequential time in seconds
PWRoots
MPSolve
Hyp Ell Flat
d=2000
d=4000
d=6000
d=8000
d=10000
d=12000
d=14000
d=16000
d=18000
d=20000

Conclusion

- New data structure for representing polynomials
- Fast numeric single and multi-point evaluation
- Fast numeric root finding
- C library soon available[1]

Perspectives

- Bivariate polynomials
- Non integer exponents
- Laplace transform



[1] https://gitlab.inria.fr/gamble/pwpoly

Thank you!