Generalized Gaussian Quadrature as a Tool for Discretizing Singular Integral Equations

James Bremer (University of California, Davis)

October 15, 2020

Joint work with Zydrunas Gimbutas (NIST Boulder) and Vladimir Rokhlin (Yale University)

A large class of linear elliptic boundary value problems can be reformulated as systems of integral equations.

Initial elliptic boundary value problem:

$$\begin{aligned} \Delta u(x) + q(x)u(x) &= f(x) \ \text{ in } \ \Omega \subset \mathbb{R}^2 \\ u(x) &= g(x) \ \text{ on } \partial\Omega \end{aligned}$$

Representation formula for the solution:

$$u(x) = \frac{1}{2\pi}\int_\Omega \log|x-y| \ \psi(y)\ dy + \frac{1}{4\pi}\int_{\partial\Omega} \frac{(y-x)\cdot\eta_y}{|x-y|^2}\ \sigma(y)\ dS(y)$$

Resulting system of integral equations:

$$\begin{aligned} \psi(x) + \frac{q(x)}{2\pi}\int_\Omega \log|x-y|\psi(y)\ dy + \frac{q(x)}{4\pi}\int_{\partial\Omega}\frac{(y-x)\cdot\eta_y}{|x-y|^2}\sigma(y)\ dS(y) &= f(x) \text{ in } \Omega \\ \frac{1}{2}\sigma(x) + \frac{q(x)}{2\pi}\int_\Omega \log|x-y| \ \psi(y)\ dy + \frac{q(x)}{4\pi}\int_{\partial\Omega}\frac{(y-x)\cdot\eta_y}{|x-y|^2}\sigma(y)\ dS(y) &= g(x) \text{ on } \partial\Omega. \end{aligned}$$

Why might one do this?

The boundary value problem

$$\Delta u(x) + q(x)u(x) = f(x) \quad \text{in} \quad \Omega \subset \mathbb{R}^2$$
$$u(x) = g(x) \quad \text{on } \partial\Omega$$

gives rise to a **well-conditioned** invertible operator $H^2(\Omega) \to L^2(\Omega) \times H^{\frac{3}{2}}(\partial\Omega)$ (for example) assuming sufficient regularity on the part of q and Ω.

This is a spatially global problem and representing its solutions locally (e.g., spectral element, finite element or finite difference methods) generally leads to an ill-conditioned discretization.

The system of integral equations

$$\psi(x) + \frac{q(x)}{2\pi}\int_{\Omega} \log|x-y|\psi(y)\, dy + \frac{q(x)}{4\pi}\int_{\partial\Omega} \frac{(y-x)\cdot\eta_y}{|x-y|^2}\sigma(y)\, dS(y) = f(x) \text{ in } \Omega$$
$$\frac{1}{2}\sigma(x) + \frac{q(x)}{2\pi}\int_{\Omega} \log|x-y|\ \psi(y)\, dy + \frac{q(x)}{4\pi}\int_{\partial\Omega} \frac{(y-x)\cdot\eta_y}{|x-y|^2}\sigma(y)\, dS(y) = g(x) \text{ on } \partial\Omega.$$

gives rise to a **well-conditioned** invertible operator $L^2(\Omega) \to L^2(\Omega) \times L^2(\partial\Omega)$.

There is no difficulty in representing the solutions ψ and σ of this system locally.

There is no free lunch ...

The discretization of systems of integral equations leads to linear systems with large, dense coefficient matrices.

Historically, the cost of solving these dense systems dominated the cost of integral equation methods and all other considerations were secondary.

But, in the low-frequency regime, this problem is "mostly solved" by analysis-based fast methods such as tree codes, fast multipole methods, and fast direct solvers.

Most of the integral operators which arise have singular kernels.

Producing accurate discretizations at a reasonable cost is challenging, especially in two and three dimensions.

Generalized Gaussian quadrature rules have emerged as one of the principal tools for doing so.

Generalized Gaussian quadrature rules on intervals

Theorem (Bojanov, Braess, Dyn): Suppose that $f_1, \dots, f_{2N} \subset C^{2N-1}[a, b]$ form an extended Chebyshev system. This means that any Hermite interpolation problem on $[a, b]$ with $2N$ degrees of freedom can be solved via a linear combination of the f_j. Suppose also that $\eta \in C\,(a, b)$ is positive. Then there exist a unique quadrature rule of the form

$$\int_a^b f(x)\eta(x)dx = \sum_{j=1}^{N} f(x_j)w_j$$

which is exact for each of the functions f_i.

A formula of this type is known as a **generalized Gaussian quadrature rule** for the functions $f_1, \dots, f_{2N}$.

But we are mostly interested in singular functions!

Ostensibly, the theorem only applies to products of fairly smooth functions with a weight function η which is possibly singular at the endpoints of the interval.

In fact, we can use various substitutions (e.g., $x = \exp(-u)$) and other tricks ($|x| = \sqrt{x^2 + \epsilon^2}$) to apply this theorem to a large class of nasty singular functions.

A numerical algorithm for constructing GGQs

Basic idea:

Apply Newton's method to solve the system

$$\int_a^b f_i(x)dx = \sum_{j=1}^{N} f_i(x_j)w_j, \quad i = 1, \ldots, 2N,$$

of $2N$ nonlinear equations in the $2N$ unknowns $x_1, \ldots, x_N, w_1, \ldots, w_N$.

This is much harder than it sounds:

As always with Newton's method, we need a good initial guess.

We are often given a large collection of **linearly dependent** functions whose span we wish to integrate.

Outline of an algorithm for the numerical computation of GGQ rules

Step one: discretize a collection of input functions $f_1, \ldots, f_M$

Step two: form an orthonormal basis $q_1, \ldots, q_K$ in the span of the input functions

Step three: construct an oversampled K-point quadrature rule

Step four: reduce the quadrature rule one point at a time using Newton's method in hope of eventually producing a rule with $\lceil K/2 \rceil$ nodes

Step one: discretize the input functions

Represent the input functions via piecewise Legendre expansions:

$$\sum_{j=1}^{m} \chi_{[a_j,b_j]}(t) \sum_{i=0}^{n} c_{ij} \sqrt{\frac{2i+1}{b_j - a_j}} P_i \left(\frac{2}{b_j - a_j} t + \frac{a_j + b_j}{a_j - b_j} \right)$$

The decomposition of the interval $[a, b]$ is determined using a heuristic procedure:

We recursively subdivide intervals until the trailing coefficients of each Legendre expansion are "small."

We expect to obtain a piecewise Gauss-Legendre rule

$$x_1, \ldots, x_l, w_1, \ldots, w_l$$

which integrates **products** of the input functions. Note that the $(n+1)$-point Gauss-Legendre rule is exact for the product of any two polynomials of degree n.

Step two: orthonormalization of the input functions

We compute a rank-revealing QR decomposition of the matrix:

$$\begin{pmatrix} f_1(x_1)\sqrt{w_1} & f_2(x_1)\sqrt{w_1} & \cdots & f_M(x_1)\sqrt{w_1} \\ f_1(x_2)\sqrt{w_2} & f_2(x_2)\sqrt{w_2} & \cdots & f_M(x_2)\sqrt{w_2} \\ \vdots & & \ddots & \vdots \\ f_1(x_l)\sqrt{w_l} & f_2(x_l)\sqrt{w_l} & \cdots & f_M(x_l)\sqrt{w_l} \end{pmatrix}$$

We use a randomized algorithm to accelerate these calculations.

The result is an orthonormal basis $q_1, \dots, q_K$ for the span of the input functions:

If S is the span of the input functions, then the map

$$f \to \begin{pmatrix} f(x_1)\sqrt{w_1} \\ f(x_2)\sqrt{w_2} \\ \vdots \\ f(x_l)\sqrt{w_l} \end{pmatrix}$$

is an isomorphism $S \subset L^2(a, b) \to \mathbb{R}^l$ and so inner products of the columns of this matrix are equal to the inner products of the input functions in $L^2(a, b)$.

Step three: construction of an initial K-point quadrature rule

We use a rank-revealing QR decomposition to select a set of K columns of the matrix :

$$\begin{pmatrix} q_1(x_1)\sqrt{w_1} & q_1(x_2)\sqrt{w_2} & \cdots & q_1(x_l)\sqrt{w_l} \\ q_2(x_1)\sqrt{w_1} & q_2(x_2)\sqrt{w_2} & \cdots & q_2(x_l)\sqrt{w_l} \\ \vdots & & & \\ q_K(x_1)\sqrt{w_1} & q_K(x_2)\sqrt{w_2} & \cdots & q_K(x_l)\sqrt{w_l} \end{pmatrix}$$

We then solve the following system to construct a K-point quadrature rule:

$$\begin{pmatrix} q_1(x_{j_1})\sqrt{w_{j_1}} & q_1(x_{j_2})\sqrt{w_{j_2}} & \cdots & q_1(x_{j_K})\sqrt{w_{j_K}} \\ q_2(x_{j_1})\sqrt{w_{j_1}} & q_2(x_{j_2})\sqrt{w_{j_2}} & \cdots & q_2(x_{j_K})\sqrt{w_{j_K}} \\ \vdots & & \ddots & \vdots \\ q_K(x_{j_1})\sqrt{w_{j_1}} & q_K(x_{j_2})\sqrt{w_{j_2}} & \cdots & q_K(x_{j_K})\sqrt{w_{j_K}} \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_K \end{pmatrix} = \begin{pmatrix} \int_a^b q_1(x)\ dx \\ \int_a^b q_2(x)\ dx \\ \vdots \\ \int_a^b q_K(x)\ dx \end{pmatrix}$$

Step four: point-by-point reduction of the quadrature rule

We remove one quadrature node from our quadrature rule:

This leaves us with a "quadrature rule" with nodes and weights

$$x_1, \dots, x_{s-1}, x_{s+1}, \dots, x_L, w_1, \dots, w_{s-1}, w_{s+1}, \dots, w_L$$

which does not accurately integrate the input functions.

We apply Newton's method to the system of nonlinear equations:

$$\sum_{\substack{j=1 \\ j \neq s}}^{L} q_i(x_j) w_j = \int_a^b q_i(x)\,dx, \quad i = 1, \dots, K$$

If Newton's method does not converge, we choose a different point and try again.

How do we decide in what order we should try to remove points?

For every node x_S, we solve the linearization of the system:

$$\sum_{\substack{j=1 \\ j \neq s}}^{L} q_i(x_j) w_j = \int_a^b q_i(x) dx, \quad i = 1, \ldots, K$$

That is, we perform **one** Newton iteration.

We order the nodes according to the l^2 norms of the solutions:

We refer to this quantity as the “significance” of the node. We try to remove nodes with low “significance” first.

A naive implementation of this approach would be extremely expensive — the asymptotic running time of the algorithm would be $\mathcal{O}\left(K^4\right)$!

These calculations can be accelerated

The linearization of the system $\sum_{j=1}^{L} q_i(x_j)w_j = \int_a^b q_i(x)dx, \quad i = 1, \dots, K$ is:

$$\underbrace{\left(\begin{array}{cccc|cccc} q_1'(x_1)w_1 & q_1'(x_2)w_2 & \cdots & q_1'(x_L)w_L & q_1(x_1) & q_1(x_2) & \cdots & q_1(x_L) \\ q_2'(x_1)w_1 & q_2'(x_2)w_2 & \cdots & q_2'(x_L)w_L & q_2(x_1) & q_2(x_2) & \cdots & q_2(x_L) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ q_K'(x_1)w_1 & q_K'(x_2)w_2 & \cdots & q_K'(x_L)w_L & q_K(x_1) & q_K(x_2) & \cdots & q_K(x_L) \end{array}\right)}_{A} \underbrace{\begin{pmatrix} \delta x_1 \\ \vdots \\ \delta x_L \\ \delta w_1 \\ \vdots \\ \delta w_L \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} \Delta_1 \\ \Delta_2 \\ \vdots \\ \Delta_K \end{pmatrix}}_{b},$$

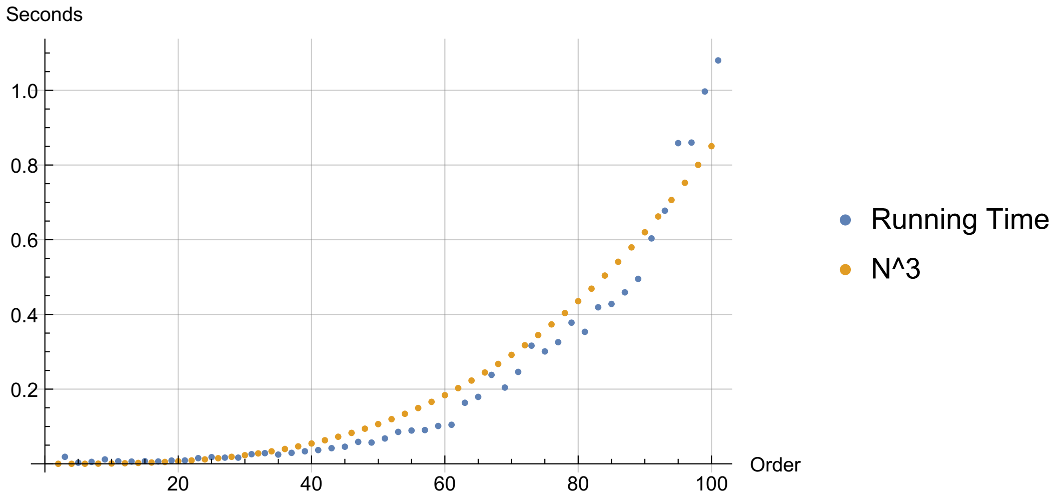
where $\Delta_i = \int_a^b q_i(x)\, dx - \sum_{j=1}^{L} q_i(x_j)w_j$.

We can find a minimum l^2 norm least squares solution by solving the normal equations:

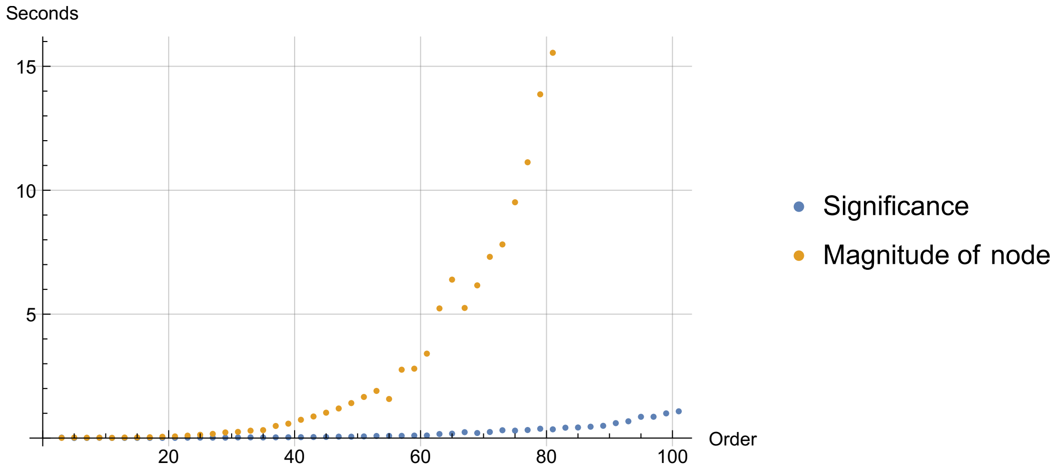
$$x = A^t(A^tA)^{-1}b$$

Deleting a node corresponds to deleting two columns of A and requires recalculating the Δ_i.

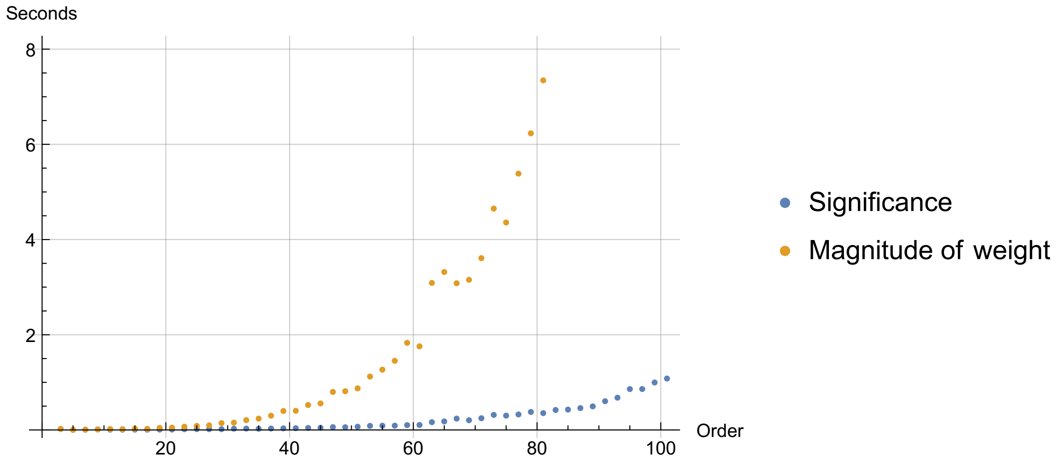
Example: $\int_0^1 \frac{P_j(x)}{\sqrt{x}}\, dx$ for all $j = 0, 1, \ldots, N$



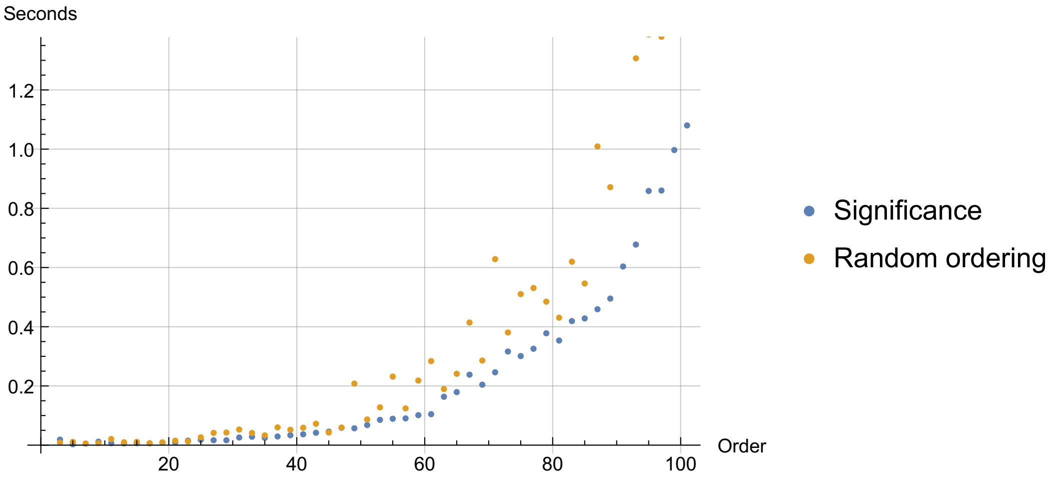
Example: $\int_0^1 \frac{P_j(x)}{\sqrt{x}}\,dx$ for all $j = 0, 1, \ldots, N$



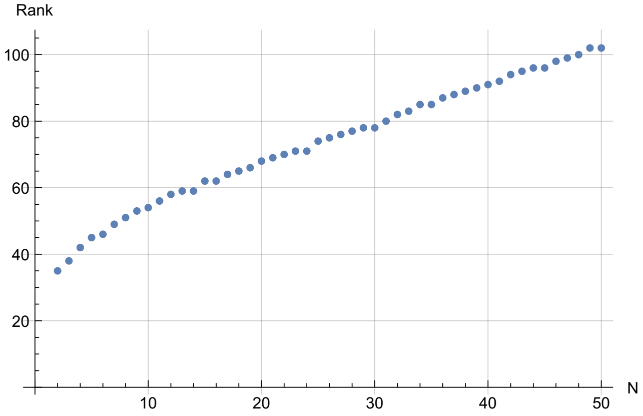
Example: $\int_0^1 \frac{P_j(x)}{\sqrt{x}}\, dx$ for all $j = 0, 1, \ldots, N$



Example: $\int_0^1 \frac{P_j(x)}{\sqrt{x}}\, dx$ for all $j = 0, 1, \ldots, N$

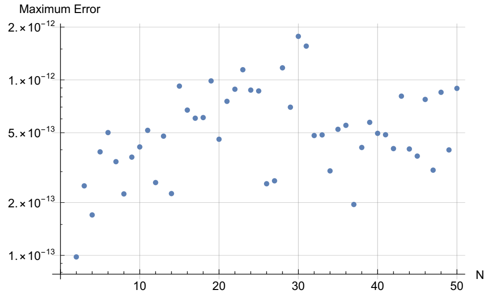


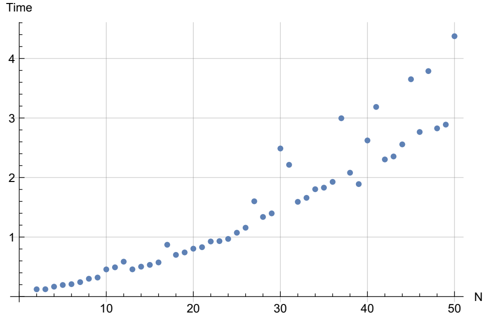
Example: $\int_{-1}^{1} |x|^{\alpha} P_j(x)dx, \quad -\frac{1}{2} \leq \alpha \leq \frac{1}{2}, \quad j = 0, \dots, N$



N	Rank	Nodes
10	54	28
11	56	29
12	58	30
17	64	33
27	76	39
30	78	40
31	80	41
37	88	45
40	91	47
41	92	47
45	96	49
47	99	51
50	102	52

Example: $\int_{-1}^{1} |x|^{\alpha} P_j(x)dx, \quad -\frac{1}{2} \leq \alpha \leq \frac{1}{2}, \quad j = 0, \ldots, N$





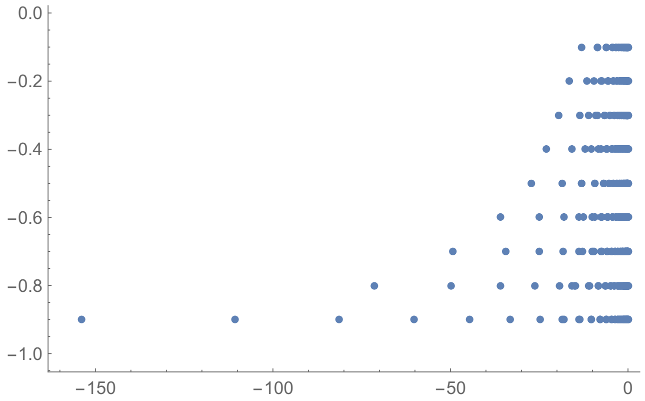
Example: $\int_{-1}^{1} \left(\log|x-y| P_i(y) + P_j(y)\right) dx, \quad x \in [-2,-1) \cup (1,2], \quad 0 \leq i,j \leq N$

Computed with 30-digit accuracy.

N	Rank	Nodes
9	129	65
14	134	67
19	138	69
24	143	72
29	147	74
39	157	79
49	168	84

Example: $\int_{-1}^{1} |x|^{\alpha} P_j(x)\,dx, \qquad 0 \le j \le 29$

Range of α	Rank	Nodes
$-0.1 \le \alpha \le 0$	66	33
$-0.2 \le \alpha \le 0$	69	35
$-0.3 \le \alpha \le 0$	72	37
$-0.4 \le \alpha \le 0$	74	38
$-0.5 \le \alpha \le 0$	76	38
$-0.6 \le \alpha \le 0$	80	41
$-0.7 \le \alpha \le 0$	83	42
$-0.8 \le \alpha \le 0$	86	44
$-0.9 \le \alpha \le 0$	93	47



Example: $\int_T r^j \theta^i \, dA$ where T is a triangle with vertices $(0,0)$, $(1,0)$, $r_0 \exp(i\theta_0)$.

These quadratures are used to discretize integral operators of the form $\int_\Sigma \frac{\sigma(y)}{|x-y|} dS(y)$ with Σ is a smooth surface.

They were constructed using the two-dimensional version of this algorithm.

The rule which holds for

$$0 \le i \le 12 \quad \text{and} \quad -1 \le j \le 12$$

$$0.9 \le r_0 \le 1.0 \quad \text{and} \quad \frac{\pi}{4} - 0.1 \le \theta_0 \le \frac{\pi}{4} + 0.1$$

has 54 nodes. The rank of the space of integrands is 152. The number of input functions was $57,000$, and the rule took approximately 30 seconds to build.

Thank you for your attention!

You can find an implementation of this algorithm (and the PDF of these slides) at:

`github.com/jamescbremerjr/GGQ`