Paraxial Approximation in CSR Modeling Using the Discontinuous Galerkin Method

D. A. Bizzozero, J. A. Ellison, K. A. Heinemann, and S. R. Lau
Department of Mathematics and Statistics, University of New Mexico; FEL 2013, New York

Introduction and Outline

- Well-established paraxial approximation in CSR has been previously modeled with a finite difference (FD) method.
- Novel approach in applying a discontinuous Galerkin (DG) method to the same equations.
- Starting point: nonhomogeneous Schrödinger type equations which arise from a paraxial approximation to Maxwell's equations.
- Include brief overview of DG and its implementation into our algorithm.
- Apply DG on a 2D transverse (x, y) grid and evolve the equations along the arclength s.
- Estimate the errors of our results and compute the longitudinal impedance.
- Discuss CPU vs GPU performance.
- Outline possible future work and new applications for DG.

Mathematical Problem

- **Domain of problem.**
  Consider rectangular domain with $(x, y) \in \Omega = [-a, a] \times [-b, b]$ and $s \in [0, L]$. Additional parameters satisfy: $k \in \mathbb{R}$ and $\rho > 0$.
- **Paraxial PDEs for $\mathbf{E}^r(x, y, s; k)$:**

$$\partial_s E_x^r = \frac{i}{2k}\nabla_\perp^2 E_x^r + \frac{ikx}{\rho}E_x^r + \frac{ikx}{\rho}E_x^b(x,y) \tag{1a}$$

$$\partial_s E_y^r = \frac{i}{2k}\nabla_\perp^2 E_y^r + \frac{ikx}{\rho}E_y^r + \frac{ikx}{\rho}E_y^b(x,y) \tag{1b}$$

$$E_s^r = \frac{i}{k}(\partial_x E_x^r + \partial_y E_y^r) \tag{1c}$$

- **Known beam component of field $\mathbf{E}^b$:**

$$E_x^b = C\frac{x}{x^2+y^2}, \quad E_y^b = C\frac{y}{x^2+y^2} \tag{2}$$

- **Boundary conditions for $E_{x,y}^r(x, y, s; k)$:**

$$\begin{aligned} \partial_x E_x^r &= \partial_y E_y^b, && \text{on} \quad x = \pm a \\ E_x^r &= -E_x^b, && \text{on} \quad y = \pm b, \end{aligned} \tag{3}$$

$$\begin{aligned} E_y^r &= -E_y^b, && \text{on} \quad x = \pm a \\ \partial_y E_y^r &= \partial_x E_x^b, && \text{on} \quad y = \pm b. \end{aligned} \tag{4}$$

- **Initial conditions for $E_{x,y}^r(x, y, s; k)$:**

$$\nabla_\perp^2 E_x^r = 0\,, \quad \nabla_\perp^2 E_y^r = 0\,, \quad \text{at} \quad s = 0 \tag{5}$$

Physical Problem

- **Beam Setup.**
  Consider a line charge moving at $v \approx c$, on a circular arc of radius ρ and length L, with perfectly conducting rectangular vacuum chamber.
- **Beam coordinate system.**
  Adopt special case with line charge reduced to a single point. Maxwell's equations written in beam coordinates (x, y, s) where the arc is in the (x, s) plane, s is the distance along the arc, and (x, y) are perpendicular to arc.
- **Electric field transformation.**
  Relate the electric field $\mathcal{E}(x, y, s, t)$ to frequency domain field $\mathbf{E}(x, y, s; k)$ by Fourier-type transformation:

$$\boldsymbol{\mathcal{E}}(x,y,s,t) \propto \int_{-\infty}^{\infty} dk \mathbf{E}(x,y,s;k)e^{ik(s-t)}$$

- **Initial condition of the electric field.**
  Assume fields reached steady-state from infinite straight prior to entering bend. Decompose electric field $\mathbf{E}$ into two components: radiation and beam field denoted by $\mathbf{E}^r$ and $\mathbf{E}^b$ respectively. $\mathbf{E}^b$ reduces the effect of the singularity.

DG Overview

- DG shares similarities with the finite element and finite volume methods.
- Rescale to dimensionless variables:

$$x \to ax, \; y \to ay, \; s \to 2ka^2 s, \; E_{x,y}^r \to Cu/a$$

- Equations (1a) and (1b) become:

$$\begin{aligned} -i\partial_s u &= \partial_x q_x + \partial_y q_y + F \\ q_x &= \partial_x u, \quad q_y = \partial_y u \end{aligned} \tag{6}$$

- Split Ω into K elements, select single element D.
- Local solution $u \in P^N(D)$.
- Multiply (6) by test functions $v \in P^N(D)$, integrate by parts over D, adjust boundary terms for fluxes (u^*, q_x^*, q_y^*), and integrate by parts again:

$$\begin{aligned} -\,i\int_D dA(v\partial_s u) &= \int_D dAv(\partial_x q_x + \partial_y q_y + F) \\ &- \int_{\partial D} dLv\big[n_x(q_x - q_x^*) + n_y(q_y - q_y^*)\big] \end{aligned} \tag{7a}$$

$$\begin{aligned} \int_D dA(vq_{x,y}) &= \int_D dAv(\partial_{x,y}u) \\ &- \int_{\partial D} dLvn_{x,y}(u - u^*) \end{aligned} \tag{7b}$$

- Expand local solution in nodal Lagrange basis:

$$u(x,y) = \sum_{j=1}^{N_p} u_j \ell_j(x,y), \quad v(x,y) = \ell_i(x,y)$$

- Vectors for nodal values $\mathbf{u} = (u_1, u_2, \ldots, u_{N_p})^T$ with $N_p = (N + 1)(N + 2)/2$.
- Similarly for $q_{x,y}$, F, ℓ, then (7a)-(7b) become:

$$\begin{aligned} -i\mathcal{M}\frac{d\mathbf{u}}{ds} &= \mathcal{S}_x\mathbf{q}_x + \mathcal{S}_y\mathbf{q}_y + \mathcal{M}\mathbf{F} \\ &- \int_{\partial D} dLn_x(\mathbf{q}_x - \mathbf{q}_x^*)\boldsymbol{\ell} \\ &- \int_{\partial D} dLn_y(\mathbf{q}_y - \mathbf{q}_y^*)\boldsymbol{\ell} \end{aligned} \tag{8a}$$

$$\mathcal{M}\mathbf{q}_x = \mathcal{S}_x\mathbf{u} - \int_{\partial D} dLn_x(\mathbf{u} - \mathbf{u}^*)\boldsymbol{\ell} \tag{8b}$$

$$\mathcal{M}\mathbf{q}_y = \mathcal{S}_y\mathbf{u} - \int_{\partial D} dLn_x(\mathbf{u} - \mathbf{u}^*)\boldsymbol{\ell} \tag{8c}$$

- Mass and stiffness matrices:

$$\mathcal{M}_{ij} = \int_D dA\ell_i\ell_j, \; (\mathcal{S}_{x,y})_{ij} = \int_D dA(\partial_{x,y}\ell_i)\ell_j \tag{9}$$

- Numerical fluxes depend on adjacent elements:

$$q_{x,y}^* = \{\{q_{x,y}\}\} - \tau[\![u]\!]_{x,y}, \quad u^* = \{\{u\}\}$$

- Full details found in *Nodal Discontinuous Galerkin Methods*, (New York: Springer, 2008) by J. Hesthaven and T. Warburton.

DG Algorithm

Step 1: Build Elements and Matrices

- Partition Ω into $K = 2N_x^{res}N_y^{res}$ triangles and space nodes optimally for matrix conditioning.

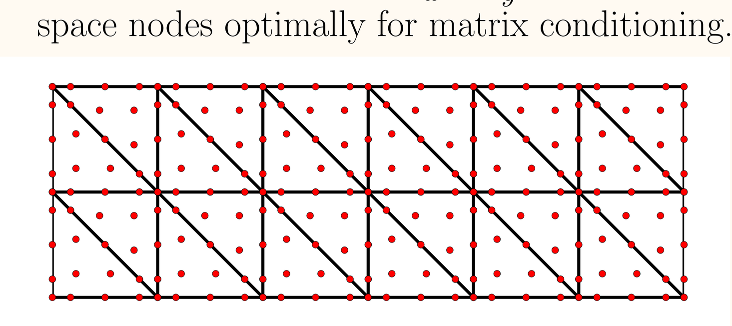

Figure 1: Mesh of $N_x^{res} = 6$, $N_y^{res} = 2$, $N = 4$.

- Build collocation derivative matrices: $\mathcal{D}_x = \mathcal{M}^{-1}\mathcal{S}_x$ and $\mathcal{D}_y = \mathcal{M}^{-1}\mathcal{S}_y$.

Step 2: Compute Initial Conditions

- Use sparse DG Poisson solver on (5) with (3), (4) to obtain initial $E_{x,y}^r$.
- Generate initial E_s^r with derivative matrices:

$$E_s^r = \frac{i}{k}(\mathcal{D}_x E_x^r + \mathcal{D}_y E_y^r) \tag{10}$$

DG Algorithm Cont.

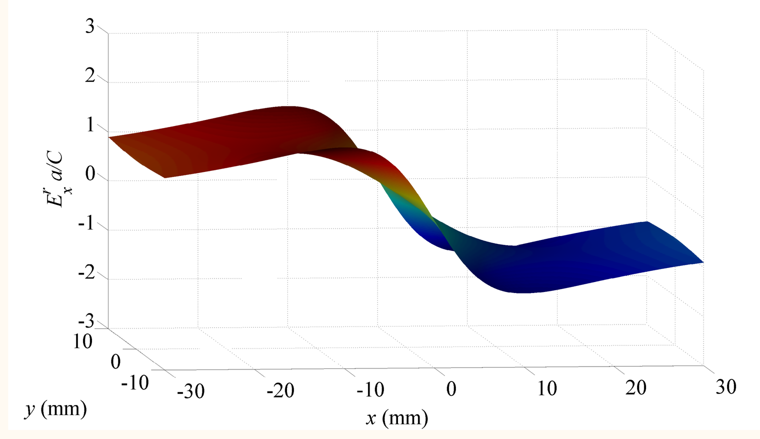

Figure 2: E_x^r initial state prior to entering bend.

Step 3: Evolve the Fields

- Estimate step size for evolution by:

$$\Delta s = C_s \cdot k \cdot r_{min}^2 \tag{11}$$

r_{min} is the minimum distance between all nodes, C_s is CFL-like constant of $\mathcal{O}(1)$. Note: $r_{min} \propto 1/(KN^2)$.

- Compute $\mathbf{q}_{x,y}$ with (8b)-(8c) and insert into (8a) for right-hand-side of $d\mathbf{u}/ds$.
- Use 4th order explicit Runge-Kutta to evolve.
- At each step, compute E_s^r with (10).

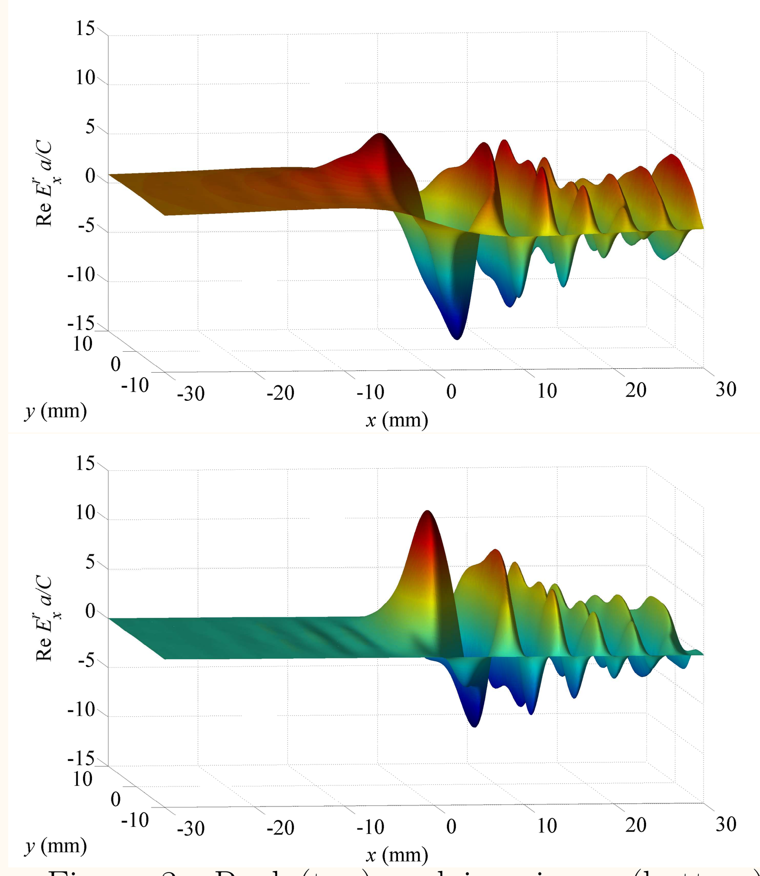

Figure 3: Real (top) and imaginary (bottom) parts of E_x^r after bend for a = 30mm, b = 10mm, L = 200mm, ρ = 1m, and k = 8mm^{-1}.

Step 4: Compute Impedance

- Evaluate impedance Z in two parts:

$$\begin{aligned} Z &= Z_b + Z_s \\ Z_b &= -\frac{Z_0}{2\pi C}\int_0^L dsE_s(0,0,s;k) \\ Z_s &= -\frac{Z_0}{2\pi C}\int_L^\infty dsE_s(0,0,s;k) \end{aligned} \tag{12}$$

- For details on the evaluation of Z, see D. Zhou's paper: Jpn. J. Appl. Phys. 51 016401 (2012).

Numerical Results: DG

DG E_x^r Error and Computation Time

N\K	150	600	1350	2400
2	8.107e-1	3.915e-1	7.471e-2	2.453e-2
	8.032e-1	2.528e-1	4.291e-2	1.484e-2
	9s	28s	49s	81s
	122	486	1093	1943
4	1.265e-1	4.897e-3	9.344e-4	3.590e-4
	8.017e-2	3.427e-3	9.462e-4	4.342e-4
	29s	88s	202s	392s
	539	2156	4850	8622
6	1.122e-2	5.177e-4	1.407e-4	5.483e-5
	6.974e-3	6.187e-4	1.932e-4	8.045e-5
	83s	283s	677s	1319s
	1691	6764	15218	27054
8	1.569e-3	1.672e-4	5.612e-5	N\A*
	1.493e-3	2.098e-4	7.473e-5	N\A*
	208s	723s	1867s	3630s
	4174	16693	37559	66771

*:Used for comparison to other tests.

Table 1: Point-wise (top), L^2 error (upper middle), computation times (lower middle), and number of timesteps (bottom) for DG code using MATLAB gpuArray implementation.

Numerical Results: FD

- FD MATLAB code is CPU-only based on Agoh and Yokoya: PR-STAB 7 054403 (2004) which uses 2nd-order stencil with leap-frog evolution.

FD E_x^r Error and Computation Time

Grid	61 × 21	121 × 41	181 × 61	241 × 81
	3.845e-1	1.126e-1	4.016e-2	3.440e-2
	4.539e-1	1.223e-1	4.551e-2	2.840e-2
	8s	125s	642s	2032s
	200	800	1800	3200

Table 2: Point-wise (top), L^2 error (upper middle), computation times (lower middle), and number of timesteps (bottom) for FD code using CPU.

Impedance Comparisons

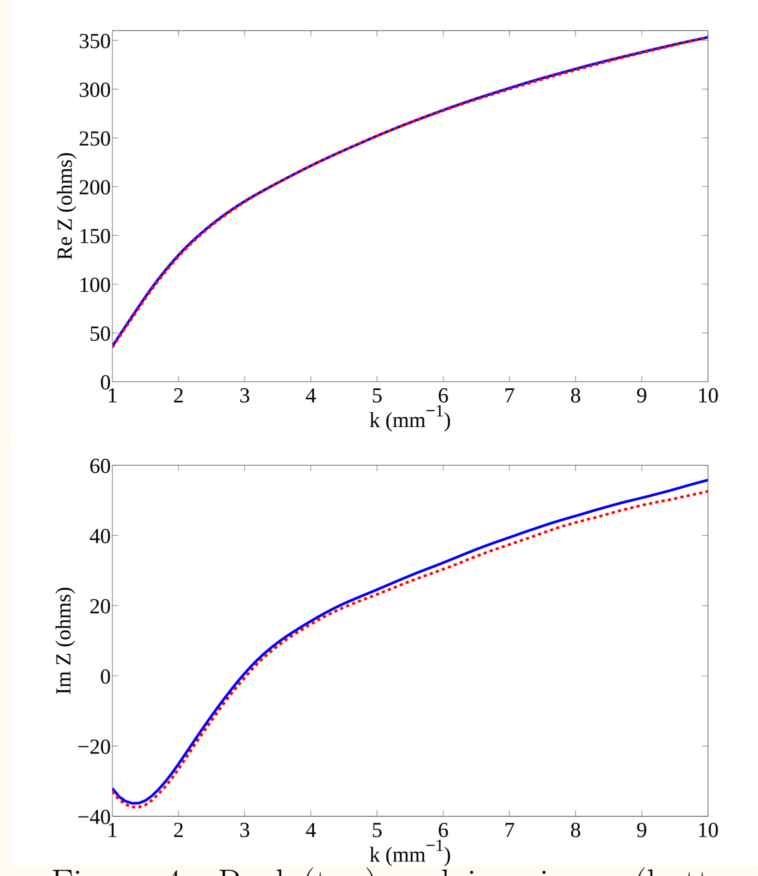

Figure 4: Real (top) and imaginary (bottom) parts of the longitudinal impedance [DG (blue solid), FD (red dashed)] for a = 30mm, b = 10mm, L = 200mm, ρ = 1m, and k = 8mm^{-1}. Agrees with D .Zhou referenced above.

GPU Computing Comments

- GPU code was adapted from CPU code with MATLAB's `gpuArray` CUDA kernel.
- MATLAB's GPU computing scales favorably for larger problems with large matrix-matrix operations or highly parallel tasks.
- Observed ∼ 60% GPU usage for high resolution run with matrices of size 45 × 2400.
- CPU : Intel Xeon E5-1620 (∼ 80Gflops/sec)
  GPU : NVIDIA GTX Titan (∼ 1.6Tflops/sec)

Future Work

- Examine spectral convergence order for DG.
- Design a higher order FD code with GPU implementation and compare performance with DG.
- Explore possible perturbation expansion of $2k^2a^3/\rho$ in rescaled versions of (1a) and (1b).
- Implement DG on Maxwell's equations without paraxial approximation and compare results.
- Consider DG on Vlasov-Maxwell's equations for future applications.

Acknowledgments

Our MATLAB DG code was built upon generic 2D DG codes written by J. Hesthaven and T. Warburton (see nudg.org). We thank T. Agoh and D. Zhou for sharing their work, D. Brewer for his timely help in making our GPU system operational, and D. Apelo for his insightful comments. Work supported by DOE under DE-FG-99ER41104.