Rapid Fourier space solution of linear partial differential equations in toroidal magnetic confinement geometries.

B. F. McMillan, S. Jolliet, A. Bottino, P. Angelino, T. M. Tran, L. Villard

Fluctuating quantities in magnetic confinement geometries often inherit a strong anisotropy along the field lines. One technique for describing these structures is the use of a certain set of Fourier components on the tori of nested flux surfaces. We describe an implementation of this approach for solving partial differential equations, like Poisson's equation, where a different set of Fourier components may be chosen on each surface according to the changing safety factor profile. Allowing the resolved components to change to follow the anisotropy significantly reduces the total number of degrees of freedom in the description. This can permit large gains in computational performance. We describe, in particular, how this approach can be applied to rapidly solve the gyrokinetic Poisson equation in a particle code, ORB5 [Jolliet *et. al.* Comp. Phys. Comm. 177, p409, (2007)], with a regular (non field-aligned) mesh.

PACS numbers:

Fourier descriptions of fluctation quantities are particularly attractive for describing waves in toroidal magnetic confinement systems: many stability[1–3] and turbulence codes[4] use a Fourier description at some point. The Fourier description is natural because of the double periodicity of the flux surfaces in magnetic confinement devices. In practice, the radial direction tends to be discretised using a finite-support scheme, and Fourier components are used to represent the variation along the toroidal and poloidal directions (which we denote χ and ζ respectively). As well as being natural, Fourier descriptions can also be quite efficient, due to the anisotropy of most of the waves of interest along the magnetic field lines. If a straight-field-line coordinate system is used, anisotropic waves can be described using a small set of Fourier components on each magnetic surface.

The usual Fourier representation of a field ϕ in magnetic devices is

$$\phi = \sum_j \sum_{(m,n)\in K} c_{j,(m,n)} \Lambda_j(s) \exp(im\chi - in\zeta), \tag{1}$$

where s is the minor radius by s, $\Lambda_j(s)$ are the finite element basis functions and K de-

notes the set of Fourier modes chosen, and $c_{j,(m,n)}$ are the discrete coefficients. The simple generalisation we suggest here is to allow K to vary with radial index j, so that the set of resolved Fourier modes varies across the radius. This has the effect of introducing a set of internal boundary conditions constraining the variation of each Fourier mode to zero outside a certain radial interval; this can be appropriate when the Fourier components of the physical modes decay rapidly to zero outside that interval. This is often the case. For example, when the modes of interest are Alfvén modes below a certain frequency, or the turbulent modes considered in gyrokinetic codes, the rapid increase in parallel wavenumber away from a rational surface leads to a localised mode amplitude.

An alternative way to efficiently represent field-aligned waves is to use a field-aligned grid. Because of the varying and non-rational twist to magnetic field lines in a typical toroidal magnetic confinement geometry, it is more complicated to implement a field-aligned grid than a regular grid in radius and toroidal and poloidal angle. For example, it can be difficult to handle the magnetic axis using a field aligned grid. However, such schemes have been succesfully implemented (for example, to treat gyrokinetic turbulence[5]). The compact support of a field aligned grid could in principle lead to a more efficient representation of the matrix problem than for a spectral representation, but for smooth equilibria the off-diagonal terms in the spectral matrix decay rapidly to zero, so that the problem is feasible in Fourier space. Code designers should be aware that a field aligned grid may be a feasible alternative to a spectral decomposition.

The bulk of this paper deals with a particular application of the Fourier representation technique: the use of the Fourier representation for the Poisson solve in the gyrokinetic code ORB5[4]. ORB5 is a particle-in-cell gyrokinetic code using a control variates method, which can handle general axisymmetric global geometries as a background equilibria. In ORB5 a grid is chosen with similar node spacing in each each dimension, of size $N_\chi \times N_\zeta \times N_s$, where N_χ, N_ζ and N_s are the grid dimensions in the poloidal, toroidal and radial directions, respectively. A filter in Fourier space is used (instead of a field aligned grid) to ensure that only the waves of interest are resolved, in order to maximise the computational timestep and reduce sampling noise. The Fourier filtering is most effective when coordinates are chosen so that the field lines are straight, so that the field lines are given by $q\chi + \zeta = \text{const}$, where q is the safety factor. The filter is applied to the perturbed ion density, and Poisson's equation is solved to determine the electric field in the plasma. Poisson's equation is discretised by

representing the density and electric potential using (up to cubic) splines in radius and toroidal and poloidal angle.

The discrete form of Poisson's equation is a matrix equation

$$\mathbf{A}\mathbf{x} = \mathbf{y} \tag{2}$$

for the gyro-density $\mathbf{y}$ in terms of the potential $\mathbf{x}$ which acts on the coefficients of the splines. Because the code solves in a axisymmetric geometry, the operator $\mathbf{A}$ is independent of the toroidal angle index, and a discrete Fourier transform is performed in toroidal angle which decouples modes of differing n. In general the matrix A is different for each n, but the standard formulation of ORB5 the toroidal derivatives are ignored, and A is the same for each toroidal mode. For each n we have a set of spline coefficients labelled by (i, j) corresponding to the grid points on the cross section (s_i, χ_j). The matrix $\mathbf{A}$ is a band matrix (the interior of the band is in general not sparse when electrons are treated as adiabatic) and the inverse operation can be evaluated using standard numerical techniques. However, the matrix is potentially very large, and the inversion can be slow and consume large amounts of memory.

Because of the field aligned filter, the number of degrees of freedom of the gyro-density vector is much smaller in Fourier space than in configuration space. Also, since the operator $\mathbf{A}$ is relatively narrow in Fourier space (the equilibrium and toroidicity terms are large only at long wavelength), the backsolved potential is also narrow in Fourier space. It therefore seems sensible to solve the matrix equation in Fourier space, discarding Fourier modes which are unimportant: that is, solving the Poisson equation in the subspace of field aligned modes. In this situation, there is no additional computational cost due to the Fourier transform, which always needs to be performed during density filtering.

We explain the algorithm for the Poisson solve in Fourier space, and why it leads to a large improvement in computational efficiency for our parallelised solver. We then present a more generic implementation of the Fourier solver using parallelised sparse matrices.

I. FOURIER TRANSFORMING THE POISSON EQUATION

Instead of representing the spatially continuous electric field and density as a Fourier series directly, we continue to represent the electric potential and gyrodensity using a spline representation. A discrete Fourier transform is then used to rapidly solve a reduced matrix

equation relating the two sets of spline coefficients. In the limit where we do not discard any Fourier modes, the formal solution to the matrix is unchanged. The approach to this limit can be used as a check of the correct implementation of the algorithm.

In Fourier space the RHS equation is written

$$y_F = FAF^{-1}x_F = A_F x_F \tag{3}$$

with $y = Fy_F$, $x = Fx_F$. We use the standard complex Fourier transforms because the resulting implementation is more elegant than for real transforms. Because Hermiticity is conserved, exact conservation of energy is possible (in the zero-timestep and infinite number of markers limit) according to Ref. [6].

In ORB5, the Fourier space density filter is still applied to the density in Poisson's equation even when we solve the matrix equation in Fourier space: we can include more modes in the subspace for solving Poisson's equation than in the density filter if desired. We refer to the projection applied in Fourier space (to the matrix and vectors) to reduce the problem size as the *matrix filter*, to distinguish it from the density filter. The matrix filter is generally wider than the density filter, depending on the computational parameters, such as how many toroidal modes must be stored per processor.

II. DETAILS OF FOURIER COMPONENTS AND POLOIDAL INDEXING

The ORB5 code constructs a separate matrix for each toroidal Fourier mode: this means that the n dependence of the matrix can be easily incorporated into the code, and simplifies parallelisation. For each toroidal mode we only need to keep weight/trial functions which are close to field aligned, with $m \in S_n = [nq(s) - \delta m, nq(s) + \delta m]$, where $q(s)$ is the safety factor, which decribes how many toroidal rotations the field line makes per poloidal rotation on each magnetic surface, and s is the magnetic surface label (the radial coordinate). δm, which is related to the maximum wavenumber resolved along the field line, is kept constant across the device so that the matrix can be handled by standard band solvers. Because q is a function of s we have a maximum and minimum m to keep on a particular flux surface for each n. The number of poloidal modes kept for each surface is then $1 + 2\delta m$. We also ensure the interval in poloidal modenumber S_n is inside the range $[-N_\chi/2, N_\chi/2 - 1]$ which are the largest and small Fourier indices: how these extreme values are handled is not usually

important because the very rapidly oscillating modes are not physically well resolved, and are typically prefiltered with a *square filter*.

Because radial Dirichlet boundary conditions are implemented in our code using a rotation in the radial spline space, we also ensure that splines in the first and last few radial positions resolve the same range of poloidal modes: $m_{\min}(i) = m_{\min}(1)$ for $i \in [1, k]$ and $m_{\min}(i) = m_{\min}(N)$ for $i \in [N_s - k, N_s]$ (for kth order basis functions). In this case the mapping between poloidal array index and Fourier index are the same for these radial positions, and the rotation in radial spline space (which is an independent rotation for each poloidal position or Fourier index) is the same as in the real-space solver.

The Poisson matrix in Fourier space can be constructed by explicitly Fourier transforming the configuration space matrix, but there is a more memory and time efficient method: performing the weak form integrations using the Fourier convolution theorem (both construction methods are implemented in ORB5). This is described in the appendix.

III. COMPUTATIONAL IMPLEMENTATION OF THE POISSON SOLVE.

To solve the Poisson equation, we first take the toroidal and poloidal complex Fourier transform of the gyrodensity spline coefficients. At this step a filter is applied to the density to select physically relevant modes. We then project into the smaller space of field-aligned modes: this is a simple restriction operation, choosing the Fourier coefficients which are inside the matrix filter. The small Fourier space problem is solved using the (precomputed) inverse matrix, and the vector components are inserted into the full Fourier space. An inverse Fourier transform is then performed.

The purpose of projecting into this space is to reduce the size of the Poisson matrix, and speed up the calculation of the inverse. Seen as a block matrix, the Fourier Poisson matrix has blocks of size $2\delta m+1$, instead of N_χ. For splines of order d, the number of super-diagonal blocks is d, and the number of non-zero elements in the inverse matrix and the memory use per matrix scales as $(d+1)(2\delta m+1)^2 N_s$ for the Fourier solver or $(d+1)N_\chi^2 N_s$ for the real-space solver. We typically have $N_\chi/\delta m \gg 1$ (100 is common for production cases), so these matrices are much smaller than the real-space matrix. We need to store N_ζ/P of these matrices on every processor, but the memory usage is typically negligible compared to that used to store the spline coefficients in real space. For the real-space solver, memory usage

can become a problem, even if the matrix is distributed across several processors.

The backsolve solution speed per toroidal mode is proportional to the number of non-zero elements in the inverse matrix (the precomputation of the matrix is not usually burdensome), so the Fourier backsolve is much faster, often by several orders of magnitude: the backsolve typically takes negligible time compared to other parts of the code. The original backsolve became dominant in overall computational time and memory demand in ORB5 for large systems, so the speedup and memory reduction are valuable.

In fact, the improved matrix solve is usually much faster than the 2D Fourier transform, which involves a parallel transpose in our parallelisation scheme. For sufficiently large problems (around grid sizes of $1024 \times 2048 \times 2048$) the number of spatial grid points (which scales like the system volume) starts to become comparable to the number of computational markers used in the PIC scheme (which scales like the cross-sectional area of the device). The consequence is that grid-based operations dominate and that memory use and the computer time per timestep start to scale like the inverse cube of ρ^* rather than $1/{\rho^*}^2$ (where ρ^* is the ratio of the ion sound gyroradius to the minor radius). For ITG simulations, the scaling starts to become non-optimal at a plasma size of about $1/\rho* \sim 1000$, around the size of ITER, which is likely to be the largest fusion device built over the coming decades. For smaller systems, resolving the dynamics in the 5-D space using the markers dominates the computation, despite the redundancy of representing anisotropic variation on an isotropic grid.

IV. USING THE FOURIER SOLVER IN STELLARATOR GEOMETRY AND MORE GENERAL PROBLEMS

The approach as detailed earlier was restricted in certain ways: it becomes inefficient for large q, and in particular, shows no improvement over a direct solution in cases where $q \to \infty$, because δm should also generally be proportional to q in order to handle field-aligned variations. Also, the analysis was restricted to axisymmetric problems where a continuous rotation symmetry was present. For problems in non-axisymmetric geometry, such as arise in stellarator physics, toroidal coupling prevents the separation of the problem in toroidal Fourier index, so that all the Fourier modes are coupled together. In Fourier space, the problem is not strictly sparse, but the terms rapidly become small far from the diagonal,

and can be ignored or treated as a correction. This implies that a sparse solver in Fourier space might be an appropriate method to solve the matrix problem.

As a proof of principle, we implemented a FEM solver which uses the transform into discrete Fourier space to construct the matrix problem, and used a parallel sparse solver, PETSc[7], to perform a backsolve. Using a sparse solver (as opposed to a banded solver) allows us to store only the coupling terms with significant strength, so that the matrix storage is not excessive even for fairly large problems. This is useful, for example, near the plasma edge, where (for tokamaks) $q \to \infty$ and many poloidal modes can interact. The resulting matrix is relatively well-conditioned because the matrix is typically diagonal-dominant: the interaction in Fourier space is short-ranged (because the equilibria vary on a long scale). This is an advantage of the Fourier technique over real-space solutions: because of the diagonal-dominance of the problem, complicated preconditioning techniques like multigrid are not necessary for iterative matrix inversion.

The algorithm involves two indexing schemes, $i \in [1, N_m]$ for the full Fourier space and $i' \in [1, N_k]$ for the restricted Fourier space. The restriction and expansion operations are very simple to program once subroutines have been built to find the mapping $i' \to i$. Matrix construction is slightly more complicated, because of the amount of indexing which burdens the code. A matrix construction routine generally loops over weight function indexes $i' \in N_r$. For each i', the coupled trial function indexes j' need to be found. One efficient way to perform this operation is to look up the full index, i, then find the radial index x_i and fourier index (m_i, n_i). We then loop over all the modes with $x_j \in x_i \pm \delta x$, $m_j \in m_i \pm \delta m(x_i)$ and $n_j \in n_i \pm \delta n(x_i)$ and add a coefficient to the matrix if it is in the restricted space. This is exactly analogous to the process needed in finite element codes with unstructured grids, where a graph of coupled nodes needs to be constructed. In practice it is often simpler or faster to set up the storage of the matrix first, by performing an initial loop over the trial and weight functions, and inserting the coefficients later (perhaps in an arbitrary order). Standard sparse matrix packages allow simple implementation of this two-step construction technique.

A moderate size test case was run on 8 processors resolving relevant Fourier modes on a $1024 \times 1024 \times 512$ grid. The physical geometry of the grid is an oblate cylinder (the cross-section is an ellipse with aspect ratio of 2), with a sinusoidal modulation to the z coordinate along the axis of the ellipse ($z' \to z + Asin(z)$ produces a coupling between the Fourier

modes in z). The Poisson equation $\nabla^2 V = U$ was represented using linear finite elements. We restricted the modes to $|m - nq| < \epsilon$, with $q(r) = 1.2 + 0.5r$, and $\epsilon = 7$. Only coupling between modes $m_i - m_j \leq 5$ and $n_i - n_j \leq 2$ was included in the matrix. For this system, we resolve 3840 resonant modes per surface. For 8 Processors, 1.7 Gigabytes of memory was used in matrix construction, and the backsolve required 200 seconds to achieve a relative accuracy of 1×10^{-5}.

V. CONCLUSIONS

The Fourier-transform solver described here allowed a massive speedup and reduction in memory resources in the spatial part of our gyrokinetic code. This kind of semi-spectral technique has been applied before: the novel ingredient here is that we choose a different set of Fourier components at each surface. This method appears to be generally applicable in codes which treat waves aligned with the magnetic fields of confinement devices and should allow performance improvements to codes which currently resolve a fixed set of Fourier components across the entire minor radius. The code changes required to implement such a solver are rather superficial because all we have performed is a projection of the problem onto a smaller space; the main burden is to provide a mapping between the full and restricted space and a means to determine which components are coupled together.

[1] B. F. McMillan and R. G. Storer, Journal of Plasma Physics **72**, 829 (2006).

[2] P. Popovich, W. Cooper, and L. Villard, Computer Physics Communications **175**, 250 (2006).

[3] D. V. Anderson, W. A. Cooper, R. Gruber, S. Merazzi, and U. Schwenn, Supercomputer **8**, 32 (1991).

[4] S. Jolliet, A. Bottino, P. Angelino, R. Hatzky, T. Tran, B. McMillan, O. Sauter, K. Appert, Y. Idomura, and L. Villard, Computer Physics Communications **177**, 409 (2007).

[5] Y. Nishimura, Z.lin, J. Lewandowski, and S.Ethier, Journal of Computational Physics **214**, 657 (2006).

[6] R. Hatzky, T. M. Tran, A. Könies, and R. Kleiber, Physics of Plasmas **9**, 898 (2002).

[7] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C.

McInnes, B. F. Smith, and H. Zhang, Tech. Rep. ANL-95/11 - Revision 3.0.0, Argonne National Laboratory (2008).

Appendix A: Convolution construction of the Fourier transformed Poisson matrix

Typical terms in the matrix $A_{(kj)(k'j')}$, with k and k' the weight and trial function radial indexes, and j and j' the poloidal indexes, can be written as a sum of w terms, each term being a product of the weight and trial functions ($\Lambda_{k,j}$ and $\Lambda_{k',j'}$ or their derivatives) and a spatially varying equilibrium function $C_w(x)$, with x the spatial position. There is also a nonlocal 'zonal flow' term (a flux surface average) due to the modelling of the adiabatic electrons which must be handled differently. In the standard case the integrations are approximated in the code via a sum over q Gauss quadrature points and N_s intervals in radius, N_χ intervals in poloidal position as

$$A_{(kj)(k'j')} = \sum_{w=1}^{n_w}\sum_{q=1}^{n_q}\sum_{K=0}^{N_s-1}\sum_{J=0}^{N_\chi-1} \tag{A1}$$

$$\frac{d^{p_s(w)}}{d^{p_s(w)}}\frac{d^{p_\chi(w)}}{d^{p_\chi(w)}}\Lambda_k(x_s)\Lambda_j(x_\chi)\Big|_{\mathbf{x}=\mathbf{x}'(q,K,J)} \tag{A2}$$

$$\frac{d^{p_s(w)}}{d^{p_s(w)}}\frac{d^{p_\chi(w)}}{d^{p_\chi(w)}}\Lambda_{k'}(x_s)\Lambda_{j'}(x_\chi)\Big|_{\mathbf{x}=\mathbf{x}'(q,K,J)} C_w\left(x(q)\right). \tag{A3}$$

Here, n_w is the number of terms in the weak form and n_q is the number of quadrature points. Note that many terms will be zero (compact support of spline functions). The Fourier transform of this is written

$$A'_{(km)(k'm')} = \frac{1}{N_\chi}\sum_{j=0}^{N_\chi-1}\sum_{j'=0}^{N_\chi-1}\exp[-2\pi\frac{imj+im'j'}{N_\chi}]A_{(kj)(k'j')} \tag{A4}$$

We now show how this can be written in a convolution form. The sums over the radius, weak form and quadrature points can be moved outside of the Fourier transform, and we only need consider the poloidal sums over the interval: we define the summand B via

$$A'_{(km)(k'm')} = \frac{1}{N_\chi}\sum_{w=1}^{n_w}\sum_{q=1}^{n_q}\sum_{K=0}^{N_s-1}\frac{d^{p_s(w)}}{d^{p_s(w)}}\Lambda_k(x_s)\frac{d^{p_s(w)}}{d^{p_s(w)}}\Lambda_{k'}(x_s)\Big|_{s=s'(q,K)} B'_{(m)(m')qwK}. \tag{A5}$$

The positions of the quadrature points can be written $x'(q,I,J) = (s[I,q], \chi_q + 2\pi J/N_\chi)$. And we have

$$B'_{(m)(m')qwK} = \sum_{j=0}^{N_\chi-1}\sum_{j'=0}^{N_\chi-1}\sum_{J} \exp[-2\pi\frac{imj+im'j'}{N_\chi}]\left.\frac{d^p\Lambda_j(\chi)}{d^px}\right|_{\chi_q+2\pi J/N_\chi} \tag{A6}$$

$$\left.\frac{d^p\Lambda_{j'}(\chi)}{d^px}\right|_{\chi_q+2\pi J/N_\chi} C_{qwK}(\chi_q+2\pi J/N_\chi). \tag{A7}$$

We also have $\Lambda_j(\chi) = \Lambda(\chi - 2\pi j/N_\chi)$ (because the spline functions are all identical displaced copies), so we can write:

$$B'_{(m)(m')qwK} = \sum_{j=0}^{N_\chi-1}\sum_{j'=0}^{N_\chi-1}\sum_{J} \exp[-2\pi\frac{imj+im'j'}{N_\chi}]\left.\frac{d^p\Lambda_j(\chi)}{d^px}\right|_{\chi_q+2\pi(J-j)/N_\chi} \tag{A8}$$

$$\left.\frac{d^p\Lambda_{j'}(\chi)}{d^px}\right|_{\chi_q+2\pi(J-j')/N_\chi} C_w(\chi_q+2\pi J/N_\chi). \tag{A9}$$

The sum indices can be cyclically permuted and the sums separated to give

$$B'_{(im)(i'm')qwI} = \sum_{j=0}^{N_\chi-1}\left.\frac{d^p\Lambda_j(\chi)}{d^px}\right|_{\chi_q+2\pi(-j)/N_\chi} exp[-2\pi imj/N_\chi] \tag{A10}$$

$$\sum_{j'=0}^{N_\chi-1}\left.\frac{d^p\Lambda_{j'}(\chi)}{d^px}\right|_{\chi_q+2\pi(-j')/N_\chi} exp[-2\pi imj'/N_\chi] \tag{A11}$$

$$\sum_{J} C_w(\chi_q+2\pi J/N_\chi) exp[2\pi i(m+m')J/N_\chi] \tag{A12}$$

$$= Z_{\mathbf{P}}(m)Z_{\mathbf{P}}(m')C'_q(m+m') \tag{A13}$$

Which is just a product of three separate 1-D Fourier transforms in j,j' and J. This can be rapidly evaluted.

The weak form of the zonal flow terms involves a product of sums:

$$Q_{(ij)(i'j')} = Z_{ii'}P_jP_{j'} \tag{A14}$$

with

$$P(j) = \sum_{J=0}^{M}\zeta_j|_{\chi_J}. \tag{A15}$$

The Fourier transform of Q can be straightforwardly expressed as

$$Q_{(im)(i'm')} = Z_{ii'}P(m)P(m') \tag{A16}$$

in terms of Fourier transforms $P(m)$ of $P(j)$. Because the spline function shape is the same for all χ points $P(j) = P$, and this reduces to

$$Q_{(im)(i'm')} = Z_{ii'} M \delta(m) \delta(m') P^2. \tag{A17}$$