NOTE:

We implement ETDSDC, IMEXSDC, and ETDRK4 schemes differently for diagonal and non diagonal linear operator Λ. Our MATLAB and Fortran codebase is thus divided into a *diagonal* and *nondiagonal* directories. We describe the difference between our two implementations for the ETDSDC scheme, though the same two methodologies can also be applied to IMEXSDC and ETDRK4 schemes.

1 Two ETDSDC$_N^M$ Implementations

We describe two different implementations for an ETDSDC$_N^M$ correction sweep: one for when Λ is a scalar or diagonal matrix and the other when Λ is a general matrix. Throughout our discussion, we let P denote the total number of spatial grid points, $\phi^k(x,t)$ denote the approximate solution, and $h\tau_i$ denote the quadrature nodes.

1.1 Λ Scalar or Diagonal Matrix

Let $\lambda_1, \lambda_2, ..., \lambda_P$ be the diagonal entries of Λ and let $\boldsymbol{\phi}^k$ be the $N \times P$ matrix

$$\boldsymbol{\phi}^k = \begin{bmatrix} \phi_1^k & \phi_2^k & \dots & \phi_P^k \end{bmatrix},$$

where $\phi_i^k = \left[\phi^k(x_i, h\tau_1), \dots, \phi^k(x_i, h\tau_N)\right]^T$ is an $N \times 1$ vector which contains the solution at the ith grid point and each of the quadrature nodes. Similarly, we define the $N \times P$ matrix

$$\mathbf{N}(\boldsymbol{\phi}^k) = \begin{bmatrix} N(\mathbf{t}, \phi_1^k) & N(\mathbf{t}, \phi_2^k) & \dots & N(\mathbf{t}, \phi_P^k) \end{bmatrix}$$

where $\mathbf{t} = [h\tau_1, h\tau_2, \dots, h\tau_N]^T$. Next, we define the $N-1 \times N$ integration matrix $\mathbf{W}^{(k)}$ which acts on the kth spatial gridpoint such that

$$\mathbf{W}_{ij}^{(k)} = \int_{h\tau_i}^{h\tau_{i+1}} e^{\lambda_k(h\tau_{i+1}-s)} L_j(s) ds, \qquad L_j(s) = \prod_{\substack{l=1 \\ l \neq j}}^{N} \frac{(s - h\tau_l)}{(h\tau_j - h\tau_l)}.$$

The matrices $\mathbf{W}_{ij}^{(k)}$ can be formed by first computing the scalars $w_{ij}(h_i\Lambda)$ as described in Section **??**, then noting that $\mathbf{W}_{i,j}^{(k)} = w_{j,i}(h_i\lambda_k)$. Next, we define the $N-1 \times P$ arrays

$$\mathbf{P0}_{ij} = \varphi_0(h_i\lambda_j) \qquad \mathbf{P1}_{ij} = \varphi_1(h_i\lambda_j).$$

We can now write an ETDSDC$_N^M$ correction sweep using MATLAB array notation:

1. $\mathbf{N} = \mathcal{N}(\boldsymbol{t}, \boldsymbol{\phi}^k)$;
2. $\mathbf{I}$ = zeros(N-1,N);
3. **for** i=1:P
4. $\quad \mathbf{I}(:,i) = \mathbf{W}(:,:,i) * \mathbf{N}(:,i)$
5. **for** i=1:P-1
6. $\quad \boldsymbol{\phi}^k(i+1,:) = \mathbf{P0}(i,:) \;.\!*\; \boldsymbol{\phi}(i,:) + \mathbf{P1}(i,:) \;.\!*\; (\mathcal{N}(h\tau_i, \boldsymbol{\phi}^k(i,:)) - \mathbf{N}(i,:)) + \mathbf{I}(i,:)$

The array $\boldsymbol{\phi}^k$ now contains the corrected solution. We note that lines 3-4 of runs slowly if implemented in MATLAB. It is possible to obtain a speed improvement if $\boldsymbol{\phi}$ is represented as a $N \times 1 \times P$ array and the MATLAB function *bsxfun* is used to perform the matrix multiplications.

1.2 Non-Diagonal Matrix Λ

We define the solution matrix $\boldsymbol{\phi}^k$ to be the $P \times N$ matrix

$$\boldsymbol{\phi}^k = \begin{bmatrix} \phi_1^k & \phi_2^k & \dots & \phi_N^k \end{bmatrix}$$

where ϕ_i^k is an $P \times 1$ vector containing the entire solution at the time-step $h\tau_i$. Similarly, we define the $P \times N$ matrix

$$\mathbf{N}(\boldsymbol{\phi}^k) = \begin{bmatrix} N(h\tau_1, \phi_1^k) & N(h\tau_2, \phi_2^k) & \dots & N(h\tau_N, \phi_N^k). \end{bmatrix}$$

The matrix coefficients $w_{ij}(h_i\Lambda)$, $\varphi_0(h_i\Lambda)$, and $\varphi_1(h_i\Lambda)$ from Sections **??** and **??** can be stored in a 4-dimensional and 3 dimensional arrays

$$\mathbf{W}(:,:,j,i) = w_{ij}(h_i\Lambda) \quad \mathbf{P0}(:,:,i) = \varphi_0(h_i\Lambda) \quad \mathbf{P1}(:,:,i) = \varphi_1(h_i\Lambda).$$

We can now write an ETDSDC_N^M correction sweep using MATLAB array notation:

1. $\mathbf{N} = \mathcal{N}(\mathbf{t}, \boldsymbol{\phi}^k)$;
2. $\mathbf{I}$ = zeros(P,N-1);
3. **for** i=1:N-1
4.   **for** j=1:N
5. $\mathbf{I}(:,i) = \mathbf{I}(:,i) + \mathbf{W}(:,:,j,i) * \mathbf{N}(:,j)$
6. **for** i=1:N-1
7. $\boldsymbol{\phi}^k(:,i+1) = \mathbf{P0}(:,:,i) * \boldsymbol{\phi}^k(:,i) + \mathbf{P1}(:,:,i) * (\mathcal{N}(h\tau_i, \boldsymbol{\phi}^k(:,i)) - \mathbf{N}(:,i)) + \mathbf{I}(:,i)$

The array $\boldsymbol{\phi}^k$ will contain the corrected solution.