**A.1.**

Consider solving the diffusion equation  over a 2D domain , which is discretized into a mesh of elements as shown in Fig. 11. At each nodal point (i.e., a vertex of an element) *A*, a discrete value  is stored; the finite dimensional approximation *qh*(**r**,*t*) of the true solution *q*(**r**,*t*) is then given by  (Eq. 36), where *NA*(**r**) is a set of basis functions.

**A.2.**

For a set of test functions *w*(**r**), the diffusion equation is recast as , where *B* denotes the boundary of , d*C* the line element for integration along *B*, and **n** the unit vector perpendicular to *B*. Setting *w*(**r**)0 at *B* and using the first-order backward finite-difference approximation , the above becomes  (Eq. 37).

**A.3.**

Taking  for *A*1,…,4 (Eq. 38), where , , , , and *x*\* and *y*\* are the *x*- and *y*-coordinates, respectively, translated in the range of [0,1] via an affine transformation, and using , Eq. (37) becomes . That Eq. (37) needs to be satisfied for any *w* (i.e., arbitrary *cB*) then leads to  which can be written as the matrix equation ***Kqh***(*t*)***f***(*t**t*) with the (*A*,*B*)-element of the matrix ***K*** given by , the *A*th-element of the vector ***qh***(*t*) given by , and the *B*th-element of the vector ***f***(*t**t*) given by . A Gauss quadrature is used to evaluate all the integrals over . The same equation holds for all the nodal points over the entire domain .

**3.1.**

Consider solving the modified diffusion equation (MDE)  over a spatial domain , which is discretized into a mesh of elements. At each nodal point (i.e., a vertex of an element) *m*, a discrete value  is stored; the finite dimensional approximation  of the true solution *q*(**r**,*s*) is then given by , where *fm*(**r**) is a set of basis functions.

For a set of test functions *w*(**r**), the MDE is recast as 

where the inner product is defined as  for two scalars *a* and *b* or  for two vectors **a** and **b**, ** denotes the boundary of , d*S* the surface element for integration over **, and **n** the unit vector perpendicular to **. Setting *w*(**r**)0 at **, the above becomes  (Eq. 24); this is the variational (or weak) form.

**3.2.**

The weak form for the approximate solution  is  (Eq. 26), where  and ; this gives . That Eq. (26) needs to be satisfied for any  (i.e., arbitrary ) then leads to the matrix equation  (Eq. 27?), where **C**, **D** and **E** are ***symmetric*** matrices with their (*m*,*n*)-element given by , , and  (*O*(*M*3) operation with *M* being the total number of nodal points?), respectively, and  is a vector with its *m*th-element given by .

**3.3.**

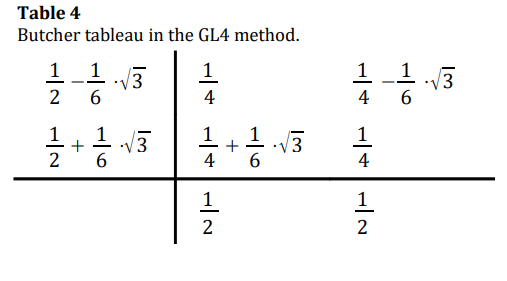
The above matrix equation can be re-written as  with . The explicit Euler’s method gives , where , *h*=1/*n* is the step size with *n* being the number of uniform subintervals (i.e., the discretization) along the chain contour, and **I** is the identity matrix; note that **C**1(**D****E**) should be computed via the LU decomposition, instead of the matrix inversion, of **C**.

Similarly, the implicit Euler’s method gives which could be tridiagonal (thus solved using the Thomas algorithm?).

To obtain more accurate results, we can use the classical 4th-order (or even higher-order) Runge-Kutta method, which gives  with , , , and .

In this work, we use the implicit 4th-order Runge-Kutta method, which gives  with  from which we solve **b**1 and **b**2 using the sparse direct solver UMFPACK (part of the SuiteSparse library).

3.4. The implicit forth order runge kutta (IRK4) is used to solve modified heat equation.



, where  and is defined as:







Find the weak form of above equation and use ,, we can find that:







This linear system is solved directly using sparse direct solver UMFPACK, which is part of the SuiteSparse library. Internally,UMFPACK uses the Unsymmetric-pattern MultiFrontal method and direct sparse LU factorization.

3.5 The boundary value of q is extrapolated using spine method with natural spline end condition.