Mario L. Gutierrez Abed
364009832
mlg3843@rit.edu

Problem Set 2
High Performance Computing

February 26, 2022

Problem 1. *The wave equation $\partial_{tt}U = \partial_{xx}U$ can be re-written into the first order form*

$$\partial_t U = K \tag{1a}$$
$$\partial_t K = \partial_x \Pi \tag{1b}$$
$$\partial_t \Pi = \partial_x K, \tag{1c}$$

where $\Pi = \partial_x U$ and $K = \partial_t U$.

Now consider the following first-order system

$$\partial_t U = {}^{u}K \tag{2a}$$
$$\partial_t {}^{u}K = \partial_x {}^{u}\Pi + \Psi(U, V) \tag{2b}$$
$$\partial_t {}^{u}\Pi = \partial_x {}^{u}K \tag{2c}$$
$$\partial_t V = {}^{v}K \tag{2d}$$
$$\partial_t {}^{v}K = \partial_x {}^{v}\Pi + \Phi(U, V) \tag{2e}$$
$$\partial_t {}^{v}\Pi = \partial_x {}^{v}K, \tag{2f}$$

where U and V are periodic with period 1 and Ψ and Φ are interaction potentials. Extend the `wave_periodic.c` *code to evolve this system. First, choose Ψ and Φ to be zero and initial conditions*

$$U = \exp\left[-10\left(1 - \cos\Theta^{-}\right)\right] \tag{3a}$$
$${}^{u}\Pi = -20\pi \sin\Theta^{-} \exp\left[-10\left(1 - \cos\Theta^{-}\right)\right] \tag{3b}$$
$${}^{u}K = 20\pi \sin\Theta^{-} \exp\left[-10\left(1 - \cos\Theta^{-}\right)\right] \tag{3c}$$
$$V = \exp\left[-10\left(1 - \cos\Theta^{+}\right)\right] \tag{3d}$$
$${}^{v}\Pi = -20\pi \sin\Theta^{+} \exp\left[-10\left(1 - \cos\Theta^{+}\right)\right] \tag{3e}$$
$${}^{v}K = -20\pi \sin\Theta^{+} \exp\left[-10\left(1 - \cos\Theta^{+}\right)\right], \tag{3f}$$

where

$$\Theta^{\pm} = 2\pi\left(x \pm \frac{1}{4}\right).$$

This should lead to a system of left and right going waves that do not interact. Use at least fourth-order accurate derivatives to implement the system. Output the values of U and V at least every

0.1 *time units and produce a simple animation showing the result. Note, if* $\Psi = \Phi = 0$*, then the exact solutions for* U *and* V *are*

$$U = e^{-10\left\{1-\cos\left[2\pi\left(x-t-\frac{1}{4}\right)\right]\right\}}, \qquad V = e^{-10\left\{1-\cos\left[2\pi\left(t+x+\frac{1}{4}\right)\right]\right\}}. \tag{4}$$

The next step is to add an interaction potential. You can choose anything you want, with the following restriction: Ψ *must be zero if* V *is zero and* Φ *must be zero if* U *is zero. We shall use*

$$\Psi(U,V) = V^2 \exp\left[-2U^2 - V^2\right], \qquad \Phi(U,V) = U^2 \exp\left[-2U^2 - V^2\right]. \tag{5}$$

Note that this system is not symmetric under exchange of U *and* V *due to the* $2U^2$ *term. Submit a report (preferably using LaTeX) containing a description of the algorithm, tests, and results. Include plots that illustrate the effects of the interaction potential. In addition, submit animations of both the wave equation with and without an interaction potential.*

Solution. The first adjustment we have to make to the code is to include the new auxiliary variables ${}^{u}\Pi$ and ${}^{v}\Pi$; thus we set `nvars=6` and now the `DECLARE_VARS` macro looks like

```
#define DECLARE_VARS(_ptr_to_gfs)\
  double __attribute__((unused))       *U       = (_ptr_to_gfs)->vars[0]->new;\
  double __attribute__((unused))      *Ku       = (_ptr_to_gfs)->vars[1]->new;\
  double __attribute__((unused))     *Piu       = (_ptr_to_gfs)->vars[2]->new;\
  double __attribute__((unused))       *V       = (_ptr_to_gfs)->vars[3]->new;\
  double __attribute__((unused))      *Kv       = (_ptr_to_gfs)->vars[4]->new;\
  double __attribute__((unused))     *Piv       = (_ptr_to_gfs)->vars[5]->new;\
  double __attribute__((unused))   *U_dot       = (_ptr_to_gfs)->vars[0]->dot;\
  double __attribute__((unused))  *Ku_dot       = (_ptr_to_gfs)->vars[1]->dot;\
  double __attribute__((unused))  *Piu_dot      = (_ptr_to_gfs)->vars[2]->dot;\
  double __attribute__((unused))   *V_dot       = (_ptr_to_gfs)->vars[3]->dot;\
  double __attribute__((unused))  *Kv_dot       = (_ptr_to_gfs)->vars[4]->dot;\
  double __attribute__((unused))  *Piv_dot      = (_ptr_to_gfs)->vars[5]->dot;\
  const int __attribute__((unused)) nsize       = (_ptr_to_gfs)->n;\
  const int __attribute__((unused)) ghost_zones = (_ptr_to_gfs)->ghost_zones;\
  const double __attribute__((unused)) dx       = (_ptr_to_gfs)->dx;
```

We then set the initial data according to Eqs. (3):

```
/* Initialization step. The user provides the initialization */
void set_initial_data(struct ngfs *gfs, const double t){

  DECLARE_VARS(gfs);

  for (int i = 0; i < nsize; i++){
    // The first "ghost_zones" points are ghostzones, x=0 occurs at index "
    ghost_zones"
    const double x          = 0 + dx * (i - gfs->ghost_zones);
```

```
    const double theta_min  = 2 * PI * (x - .25);
    const double theta_plus = 2 * PI * (x + .25);

    // Eqs 3a - 3f on Latex doc
    U[i]   = exp(-10 * (1 - cos(theta_min)));
    Piu[i] = -20 * PI * sin(theta_min) * exp(-10 * (1 - cos(theta_min)));
    Ku[i]  = 20 * PI * sin(theta_min) * exp(-10 * (1 - cos(theta_min)));
    V[i]   = exp(-10 * (1 - cos(theta_plus)));
    Piv[i] = -20 * PI * sin(theta_plus) * exp(-10 * (1 - cos(theta_plus)));
    Kv[i]  = -20 * PI * sin(theta_plus) * exp(-10 * (1 - cos(theta_plus)));
  }
}
```

With the initial conditions now set, we allocate memory for the potential functions Ψ and Φ, and then execute our main loop:

```
  // allocate memory for the potential
  double* Psi = calloc(n, sizeof *Psi);
  double* Phi = calloc(n, sizeof *Phi);

  // Make sure the memory has been successfully  allocated
  if ( (Psi == NULL) || (Phi == NULL) ) {
    printf("\tMemory not allocated. Perhaps n is too large...\n");
    return EXIT_FAILURE;
  }

  const double time_interval = .1;

  for (; t <= 30;){
  /* first update values of potentials
  *  (last arg is 0 if we have a vanishing potential,
  *  otherwise we set it to 1...see function implementation)
  */

    update_potential(&gfs, Psi, Phi, 0);

    // then take a time-step with RK4
    RK4_Step(&gfs, t, dt, Psi, Phi, wave_eq_time_deriv);
    t += dt;

    // output every 0.1 time units
    if (t / time_interval == t*10)
      output_gfs(&gfs);
  }
```

Now let us give more details about the functions being called inside this loop. The `RK4_Step` routine is essentially unchanged, except for the fact that we now also pass Ψ and Φ as arguments; this is required by the `wave_eq_time_deriv` function that is being passed to `RK4_Step`.

Let us take a look at the latter here:

```
// Calculate the time derivative of the evolved variables:
void wave_eq_time_deriv(struct ngfs *gfs, double* psi_fun, double* phi_fun, const
    double t0){

  DECLARE_VARS(gfs);

  const double h = dx;
  const double oo12h = 1. / (12. * h);

  for (int i = ghost_zones; i < nsize - ghost_zones; i++){
    U_dot[i] = Ku[i];
    V_dot[i] = Kv[i];
  }

  for (int i = ghost_zones; i < nsize - ghost_zones; i++){
    /* 4th-order centered stencil for first-derivatives */
    const double Piu_x = D4CEN(Piu, i, 1, h);
    const double Piv_x = D4CEN(Piv, i, 1, h);
    const double Ku_x  = D4CEN(Ku,  i, 1, h);
    const double Kv_x  = D4CEN(Kv,  i, 1, h);

    Piu_dot[i] = Ku_x;
    Piv_dot[i] = Kv_x;
    Ku_dot[i]  = Piu_x + psi_fun[i];
    Kv_dot[i]  = Piv_x + phi_fun[i];
  }
```

We see that Ψ and Φ are passed in order to fulfill Eqs. (2b) and (2e). The routine `D4CEN` provides a centered scheme with fourth-order accuracy for first-derivatives:

```
#define D4CEN(f, i, di, h) \
  ((f[-2 * di + i] - 8 * f[-di + i] + 8 * f[di + i] - f[2 * di + i]) * oo12##h)
```

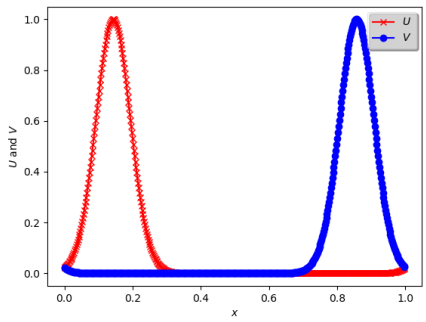
Now all that's left to explain is the `update_potential` routine. As remarked in the code, we set the value of the last argument to 0 if we want a zero potential; otherwise if we want to use the potential as given by Eq. (5), we set this value to 1:

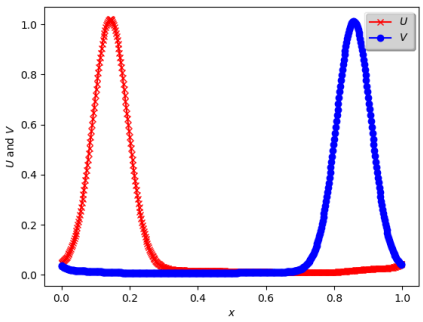
```
void update_potential(struct ngfs *gfs, double* psi_fun, double* phi_fun, size_t
    potential){
  /* The argument 'potential' is set to zero if we want a vanishing potential.
   * Otherwise we set it to 1 if we want the potentials given
   * by Eq. (5) from the Latex report.
  */
    DECLARE_VARS(gfs);

    for (int i = 0; i < nsize; i++){
        switch(potential){
```

```
            case 0:{
              psi_fun[i] = 0.;
              phi_fun[i] = 0.
            }
            break;
            case 1:{
                psi_fun[i] = V[i] * V[i] * exp(- 2. * U[i] * U[i] - V[i] * V[i]);
                phi_fun[i] = U[i] * U[i] * exp(- 2. * U[i] * U[i] - V[i] * V[i]);
            }
            break;
            default:{
                psi_fun[i] = 0.;
                phi_fun[i] = 0.;
            }
        }
    }
}
```

However, I did not find any noticeable difference between the zero- and non-zero potential cases. The values of the potential functions in the non-trivial case are still quite close to 0, so the difference they make is negligible. Here in the figure we see a snapshot of the waves at the end of our time evolution; on the left figure we used a vanishing potential, while on the right we have a nontrivial potential:





A movie for the two cases is also being uploaded in this assignment. Again, no difference is spotted between the two cases. ♠