PCHIP Interpolation

CAP-418 Numerical Methods I

2023-05-09

Contents

Objectives

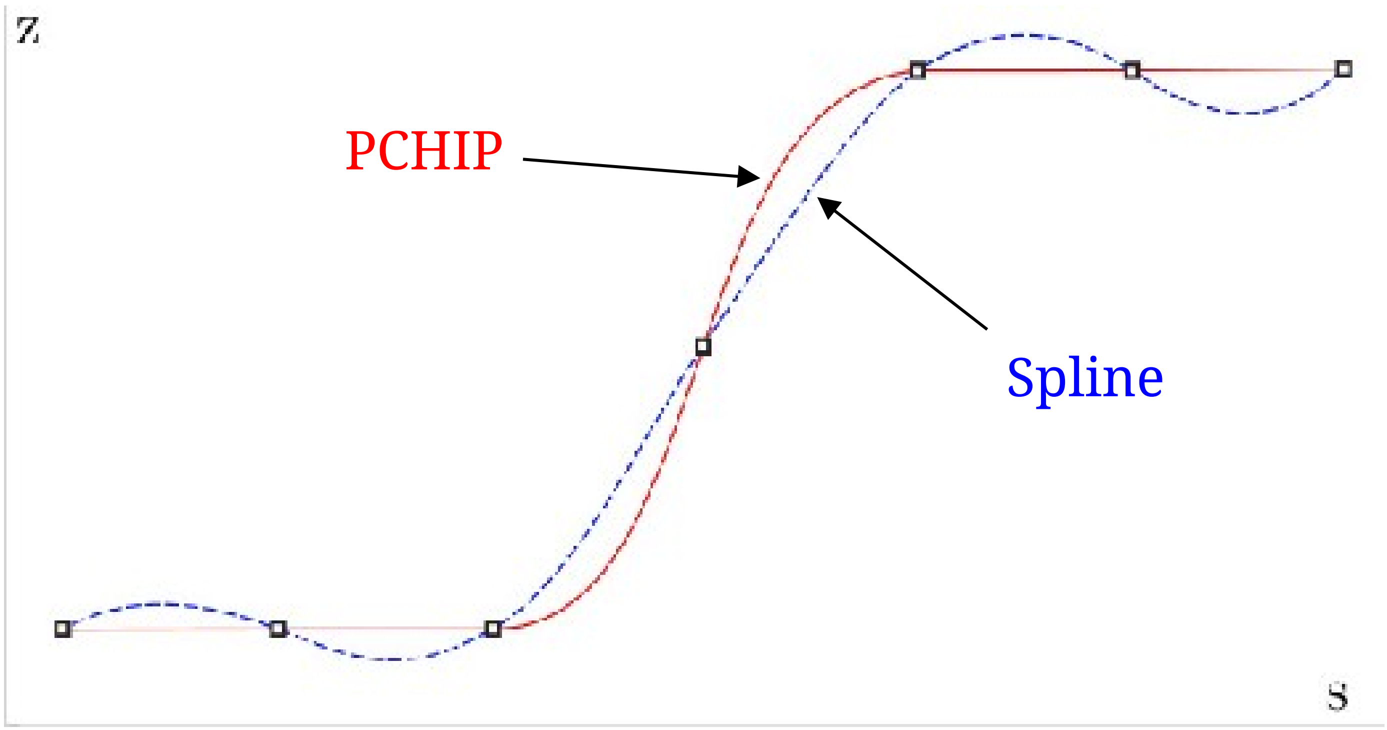
- Discretization and resampling of functions by bootstrap: PCHIP method
- Application of the PCHIP method to simulate the 1D Burgers equation
- Compare with Gauss-Hermite Quadrature method (GHQ)
- A second phase of the project (work in progress) foresees the use of GPA, 2D Burgers, and PINN

Gauss-Hermite Quadrature rule (GHQ)

- A Gaussian quadrature form for approximating integral values
- The solution of the Burgers equation use estimated values calculated using GHQ, as described in the Basdevant et al. (1986) article *Spectral and finite difference solutions of the Burgers equation*
- The implementation is based on that of John Burkardt https://people.sc.fsu.edu/~jburkardt/

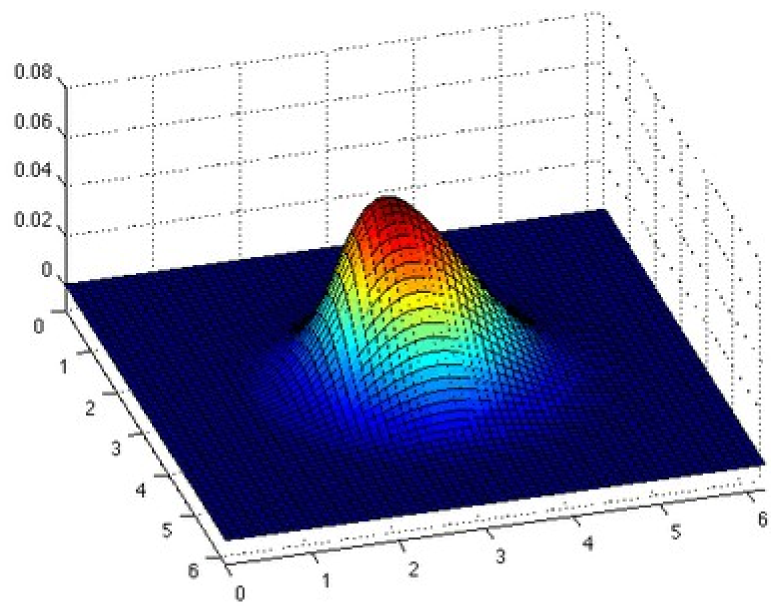
Shape-preserving piecewise cubic interpolation (PCHIP)

- PCHIP takes the same form as the GHQ, but with the first derivatives defined in a special way
- The new function values do not exceed the function values at the end of each range



Burgers' equation toy problem

- Is a fundamental partial differential equation and convection–diffusion equation
- GHQ uses Burgers to generate a dataset
- This dataset is then used in PCHIP



$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

for -1.0 < x < +1.0, and 0 < t

I.C. $u(x,0) = -\sin(\pi x).$

B.C. $u(-1,t) = u(+1,t) = 0$

Viscosity $\frac{0.01}{\pi}$

GHQ implementation

- While some algorithms are available, this work has not focused on GHQ

Algorithm 1 Algorithm to find approximate solutions of the Burgers' equation by splitting methods (11) using (15).

for i=1 to p+1 **do**
  for j=0 to M **do**
    Solve the iterative method (17) by taking time step as $k = a_i k$ for given initial condition.
- Set $U^{(n)}_{m,j+1} = U^{(n_j)}_{m,j}$ for all m
- **while** condition (18) satisfy **do**
- Compute values of $U^{(n+1)}_{m,j+1}$ from (17)
- Take $U^{(n)}_{m,j+1} = U^{(n+1)}_{m,j+1}$ for all m
- Compute improved values of $U^{(n+1)}_{m,j+1}$ from (17)
- **end while**

      if $b_i \neq 0$ **then**
        Take the computed values of $U^{(n+1)}_{m,j+1}$ as initial conditions of formula (19) and solve it by taking time step as $k = b_i k$ to find values $U_{m,j+1}$.
      end if
    end for
  end for

GHQ implementation

- The implementation is adapted from the work of J. Burkardt
- Uses an algorithm by Elhay and Kautsky
- The abscissas are the zeros of the N-th order Hermite polynomial
- The integral and the quadrature rule are given by

$$\int_{-\infty}^{+\infty} e^{-x^2} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

GHQ implementation

Implementation uses Python and Jupyter Notebook

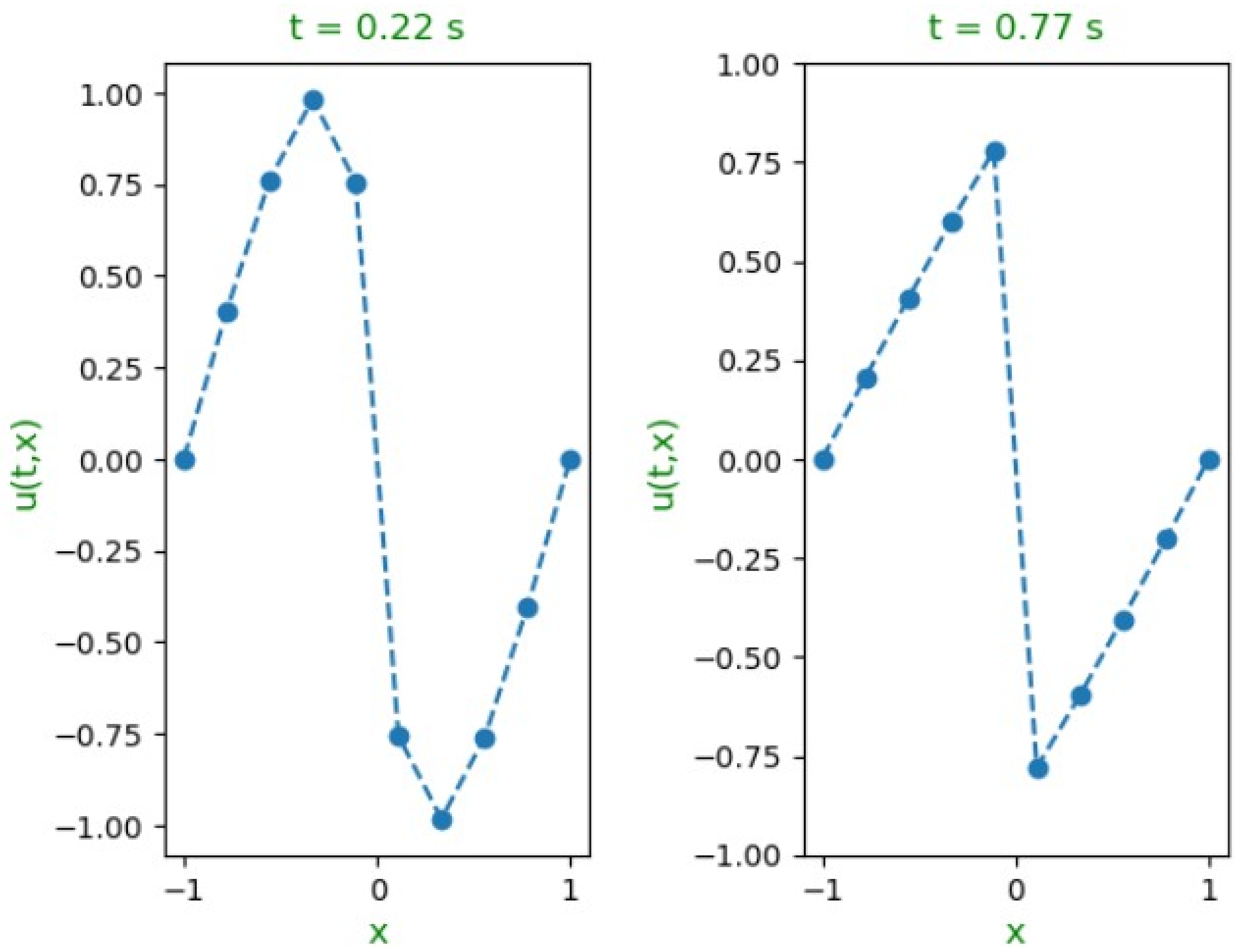
A part of the code is shown below:

```
def burgers_viscous_time_exact(nu, vxn, vx, vtn, vt, qn):
    import numpy as np

    qx, qw = hermite_ek_compute(qn)
    #  Evaluate U(X,T) for later times.
    vu = np.zeros([vxn, vtn])
    for vti in range(0, vtn):
        if (vt[vti] == 0.0):
            for i in range(0, vxn):
                vu[i, vti] = -np.sin(np.pi * vx[i])
        else:
            for vxi in range(0, vxn):
                top = 0.0
                bot = 0.0
                for qi in range(0, qn):
                    c = 2.0 * np.sqrt(nu * vt[vti])
                    top = top - qw[qi] * c * np.sin(np.pi * (vx[vxi] - c * qx[qi])) \
                        * np.exp(- np.cos(np.pi * (vx[vxi] - c * qx[qi]))
                                 / (2.0 * np.pi * nu))
                    bot = bot + qw[qi] * c \
                        * np.exp(- np.cos(np.pi * (vx[vxi] - c * qx[qi]))
                                 / (2.0 * np.pi * nu))
                    vu[vxi, vti] = top / bot
    return vu
```

Parameters

```
import numpy as np

vtn = 10  # NT : Time t
vxn = 10  # NX : Variable x
nu = 0.01 / np.pi  # Viscosity
qn = 50  # Quadrature order

xlo = -1.0
xhi = +1.0
vx = np.linspace(xlo, xhi, vxn)

tlo = 0.0
thi = 0.99  # other option: thi = 3.0 / np.pi
vt = np.linspace(tlo, thi, vtn)

vu = burgers_viscous_time_exact(nu, vxn, vx, vtn, vt, qn).T
```

Result



PCHIP interpolation

- Uses SciPy and the small dataset generated by GHQ

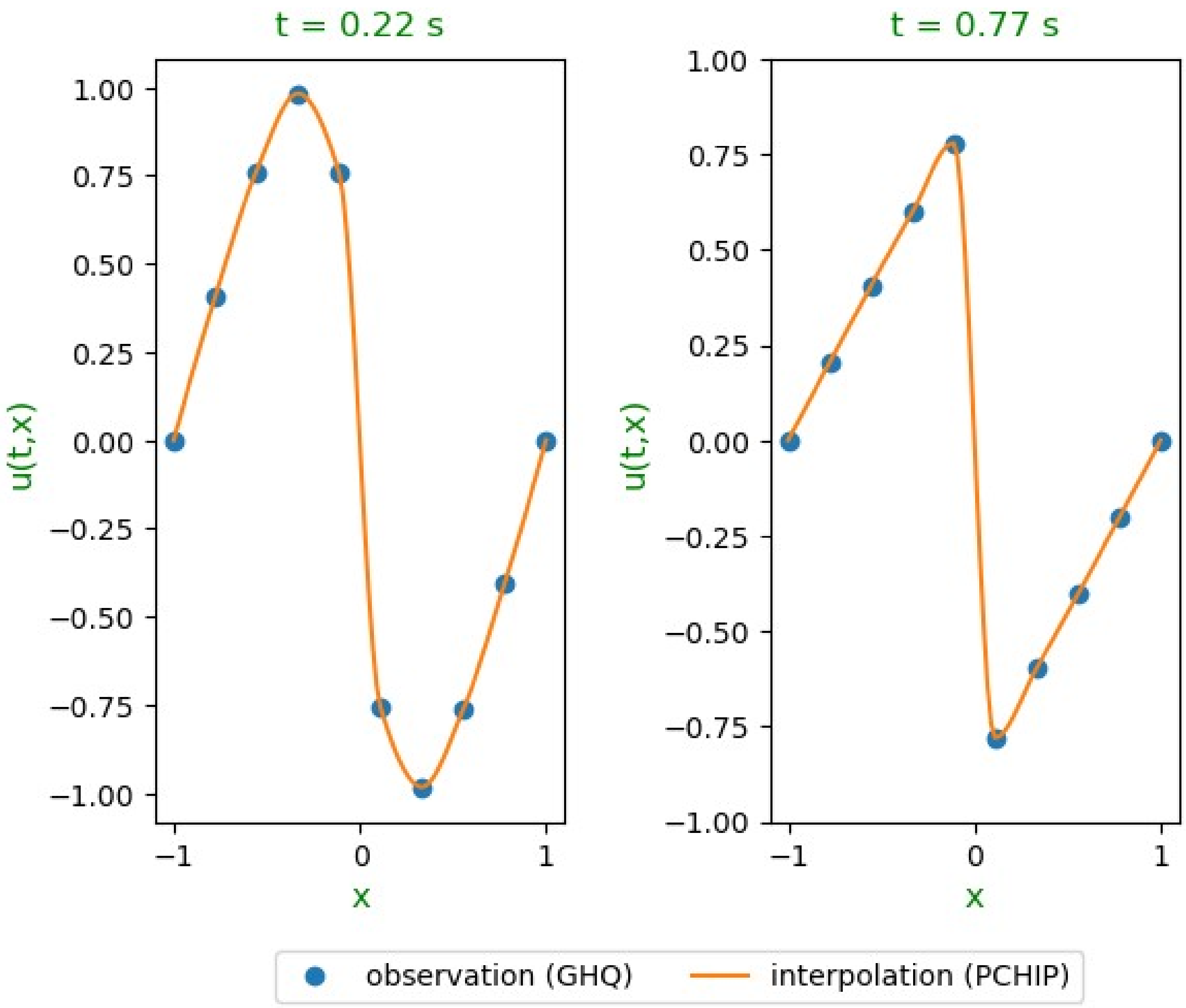
```
from scipy.interpolate import pchip_interpolate
```

- The points specified in xq (query points) are the x-coordinates for the interpolated function values yq computed by pchip :

```
x_observed = vx
y_observed = vu[2, :]
xq = np.linspace(min(x_observed), max(x_observed), num=100)
yq = pchip_interpolate(x_observed, y_observed, xq)
```

Result

Using a small dataset of 10 points

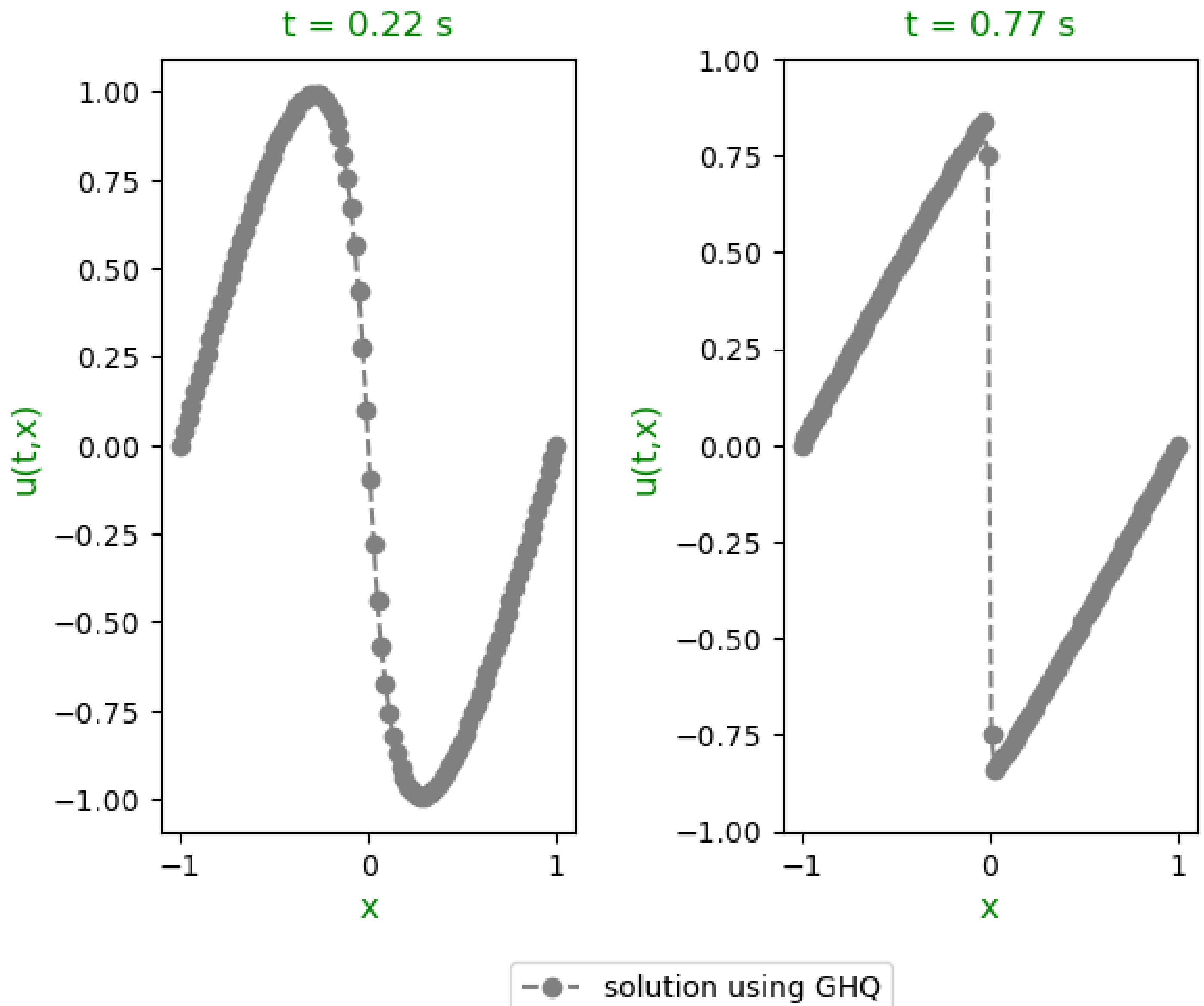


1D Burgers solution using GHQ

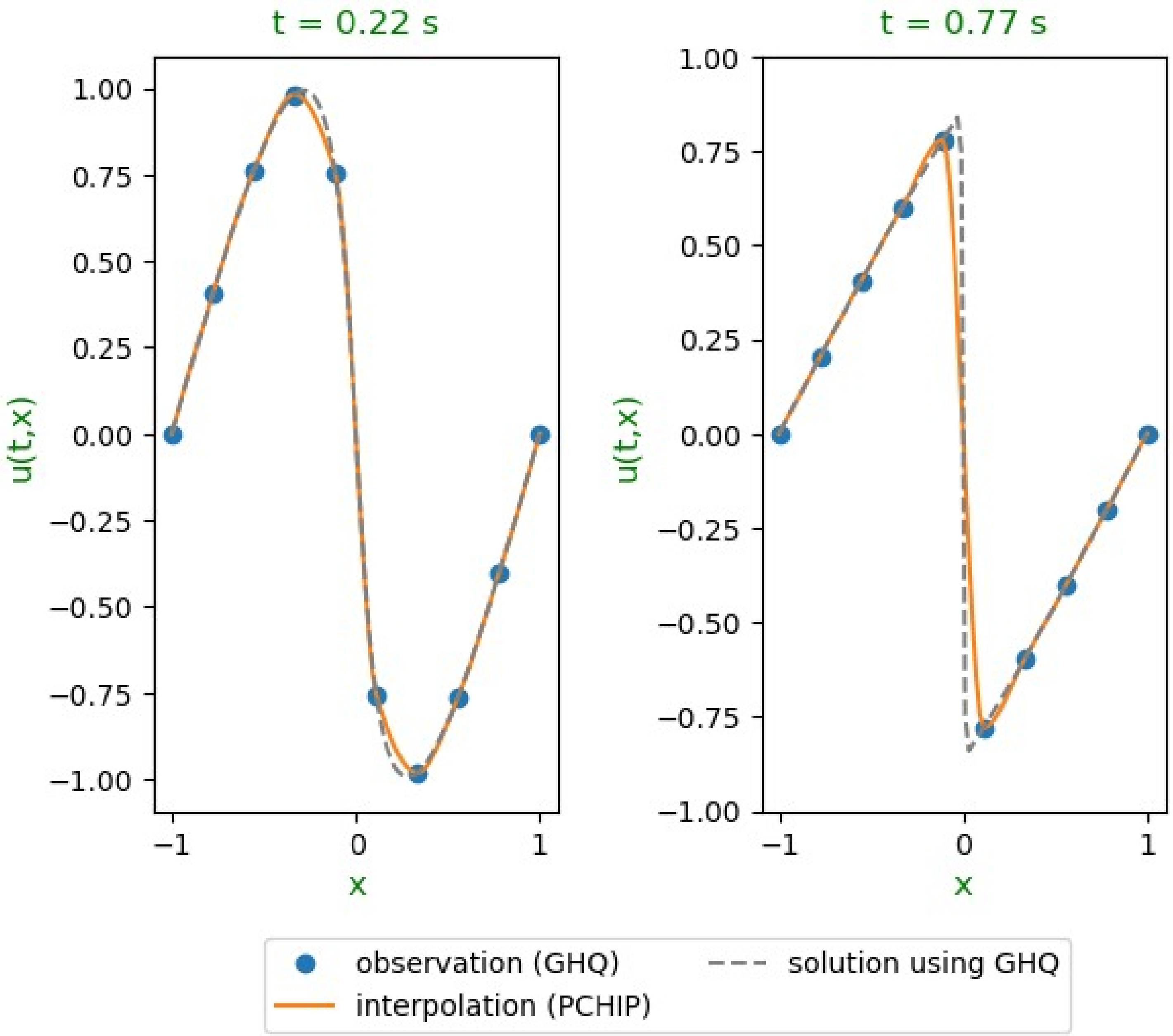
- Generates a dataset containing 100 points

```
vtn2 = 100  # NT : Time t
vxn2 = 100  # NX : Variable x
nu = 0.01 / np.pi  # Viscosity
qn = 50  # Quadrature order

xlo = -1.0
xhi = +1.0
vx2 = np.linspace(xlo, xhi, vxn2)

tlo = 0.0
thi = 0.99  # other option: thi = 3.0 / np.pi
vt2 = np.linspace(tlo, thi, vtn2)

vu2 = burgers_viscous_time_exact(nu, vxn2, vx2, vtn2, vt2, qn).T
```

Only CHG


t = 0.22 s
t = 0.77 s
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
u(t,x)
−1
0
1
x
solution using GHQ

Comparison of PCHIP and GHQ curves



Conclusion

- This work compared the PCHIP interpolation method with the Gaussian quadrature method, using the solution of the 1D Burgers equation as a toy problem
- The Gauss-Hermite quadrature method was used both to generate the points that were interpolated and to generate the solution that is compared with PCHIP
- The result shows that there is a certain special deviation near the sharp edges of curves at local maxima or minima. However, the PCHIP method generally showed good performance to interpolate the selected toy problem

Future works

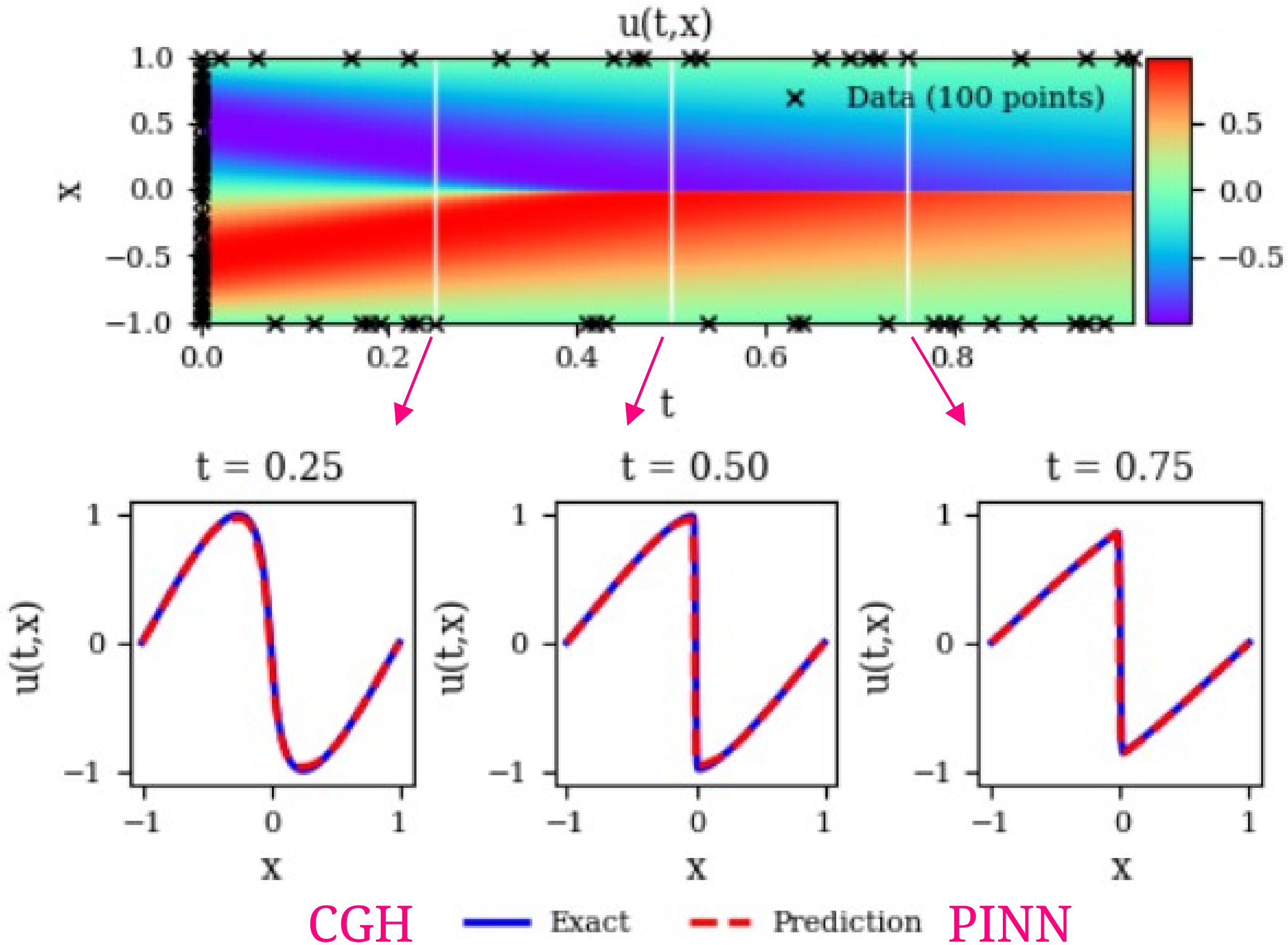
- PCHIP implementation using the algorithm (not the library)
- 2D Burgers
  - Rewrite the work
- Gradient Pattern Analysis (GPA)
  - https://github.com/rsautter/GPA (Rubens Sautter)
- PINN (work in progress, see next slide)
- Compare the performances

PINN (work in progress)

https://github.com/efurlanm/418/blob/master/burgers-tf-02.ipynb

Plot

```
plot_inf_cont_results(X_star, u_pred_flat, X_u_train, u_train, Exact_u, X, T, x,
                      t)
```



- Burgers
- CHG
- PINN
- TensorFlow v2
- Running on GPU

Thanks

https://github.com/efurlanm/418