UNDER REVIEW

RESCIENCE C

Replication / Physics

[Rp] Reproduction of Step width enhancement in a pulse-driven Josephson junction

Sabino Maggi[1]
[1]National Research Council, Institute of Atmospheric Pollution Research, CNR-IIA, Bari, Italy

Edited by
Nicolas P. Rougier

Reviewed by
Pierre de Buyl

Received
01 March 2020

Published
23 June 2020

DOI
–

1 Introduction

A Josephson junction is a quantum mechanical device composed of two superconducting electrodes separated by a weak link [1]. For currents lower than a critical value I_C, coupled electrons (Cooper pairs) can cross the weak link without a potential difference (dc Josephson effect). When the current is increased above I_C, single electrons originated by the breakup of Cooper pairs begin to traverse the weak link. The potential difference V between the two superconducting films becomes $\neq 0$ and a state is reached where the junction behaves as a resistance.
In modern Josephson junctions the weak link is usually a thin insulating tunnel barrier (SIS junction) [2], a normal metal film (SNS junction) [3] or a physical nanoconstriction (ScS junction) [4, 5]. Josephson junctions have found wide usage in several research fields, for example as building blocks for RSFQ digital electronics or quantum computers [6], or as radiation detectors and very sensitive magnetometers (SQUIDs) [7, 8, 9]. But the most successful application of Josephson junctions is surely in voltage metrology. A microwave radiation of frequency f can phase lock the junction oscillations, producing the so-called Shapiro-steps, i.e., current steps at the quantized voltages V_n,

$$V_n = n\frac{h}{2e}f, \quad n = 1, 2, \ldots \tag{1}$$

where h and e are the Plank constant and electron charge, respectively. The ac Josephson effect is at the basis of the current quantum voltage standard.
Besides its practical applications, the Josephson junction is important from a physical point of view because it has been the first device showing a quantum mechanical effect on a macroscopic scale.
An important research topic at the beginning of the '90s was related to finding ways to increase the amplitude of the current steps induced by the microwave radiation (rf-induced steps). In fact, the stability of the lock between the phase of the junction and the applied microwave radiation – and therefore its insensitivity to noise events which might switch the junction from one quantized voltage to another, a crucial problem for voltage standard applications – is strongly dependent on the amplitude of the steps [10]. To increase the amplitude of the current steps, a non-sinusoidal microwave radiation may be used. In 1990 Monaco showed that, in the limit of a voltage-biased Josephson

junction, adding together two phased microwaves of frequency f and $2f$ produces rf-induced current steps whose amplitudes are larger than those observed with a sinusoidal radiation [11].
Experiments on the so-called "biharmonic drive" readily confirmed these conclusions, albeit with some limitations due to the fact that the junctions could not realistically be considered as voltage biased [12, 13].
Extending further the idea, Monaco showed that, still in the limit of voltage bias, if the microwave radiation is composed of a train of delta functions, the rf-induced current steps could become as large as the critical current I_C.
However, a voltage bias configuration does not properly model a real Josephson junction, which should usually be considered as current biased. Also, a pulse train composed of delta functions is only a theoretical approximation and cannot be reproduced in actual experiments. This led to the idea to investigate what happened to a current-biased Josephson junction irradiated by a more realistic pulsed microwave signal [14, 15]. The reproduction of this investigation is the object of the present work.

2 Computational context

A first attempt to solve this problem was made using an electronic analog simulator of a Josephson junction [16], that could compute the relation between the applied current and the voltage ($I - V$ characteristic) of a current-biased junction, in the framework of the Stewart-McCumber RSJ junction model [17, 18]. The analog simulator was fast and simple to use, and could produce in just a few minutes on a Hewlett Packard 7475A2 pen plotter beautiful plots of the $I - V$ characteristics of the junction as a function of the simulated microwave signal. [1]
However, even if the electronic simulation was extremely fast, the analysis of the results required to measure by hand the amplitude of the rf-induced current steps visible on each $I - V$ characteristic, a tedious and error-prone task.
I then decided to develop a Fortran program to solve numerically the nonlinear second-order differential equation that models the Josephson junction [17, 18]. The idea was to calculate the $I - V$ characteristics of the junction as a function of the amplitude of the microwave signal, α_{rf}, for a given set of parameters characterizing the junction and the microwave, considering the three different cases of standard sinusoidal drive, biharmonic drive and pulsed drive.
To ease comparison of the results, normalized units were used throughout the calculations. The normalized junction voltage was η and the normalized current α_{dc}. The main parameters of the simulation were: hysteresis parameter β, microwave frequency Ω, amplitude of the microwave signal α_{rf}, pulse width ρ, integration time τ. For a given set of junction parameters, usually 100 different $I - V$ characteristics for increasing or decreasing values of α_{rf} were calculated.
The first versions of the Fortran program were compiled under DOS 6.22 with Microsoft Fortran 5.1 and run on what was then a state-of-the-art PC, probably a Compaq Deskpro 486 with a math coprocessor, shared among several users of the lab.
The limitations of a PC for such a task soon become evident. A new simulation started automatically each evening and took the entire night to complete. People still using the PC late in the evening often inadvertently stopped the background process or simply shut the machine down without checking if there was another job running. At the end, I could run a full simulation only every two or three days.
After a few weeks of these mostly unsuccessful attempts, a colleague of another research group proposed me to use four DEC workstations running ULTRIX for my own simula-

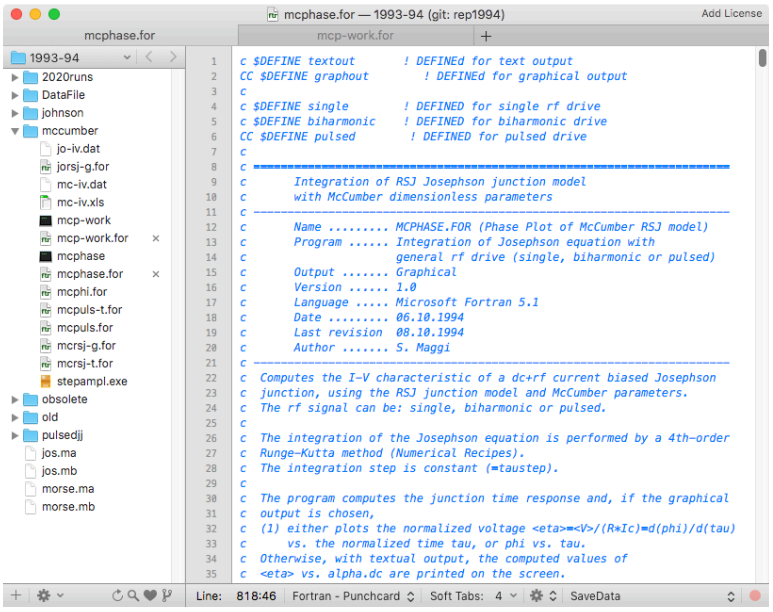
[1] Unfortunately, after 25 years and two relocations, I could not manage to find photographs of the simulator nor the original HP 7475A plots.

tions.[2] The machines were heavily used by his group during working hours, but sat mostly idle overnight. If I could manage to finish my runs before the start of the new work day, I was allowed to use this idle time for my own simulations. The colleague gave me a quick crash course on Unix and I was ready to go.
Porting my Fortran program from Microsoft Fortran 5.1 to ULTRIX was a breeze, and I quickly learnt how to use FTP to transfer the input configuration files and the output data files containing the results of the simulations back and forth from the DEC workstations to my Compaq 386 notebook, that was also my desktop computer.
Now each day I had four different sets of data files coming from the DEC workstations. Three of them were obtained by irradiating the junction with a pulsed drive with decreasing values of ρ, while keeping constant β and Ω. The fourth simulation was made by irradiating the junction with a sinusoidal drive, while keeping everything else equal. This last simulation was used as a reference, to compare the results obtained with a standard sinusoidal radiation with those obtained with progressively shorter pulses.
Each night I changed the values of β or Ω, to study the effect of these parameters on the behavior of the junction.
Again, the real problem was how to analyze all this data. A manual analysis like that needed with the electronic simulator was out of consideration. I decided to try the recently released Microsoft Visual Basic 1.0 for Windows, writing another program that calculated the size of the rf-induced current steps visible on the $I - V$ characteristics of the nightly simulations, as a function of α_{rf}.
The results of months of calculations were summarized in a paper published in the Journal of Applied Physics [14].

3 Digging into code

I like organisation, and I try keep all my past projects on my main workstation. Thus, finding the original source and data files of this project was only a question of locating the directory where the project was stored. Problems started to arise when I looked at the different files. The whole project was scattered into several directories, each containing many files with widely different names and dates. At first, trying to find an order in that chaos seemed impossible.
Normally I would have found all information needed in many notebooks full of detailed handwritten notes. Unfortunately, a couple of years ago most of my work notebooks were damaged by a water leak in the basement, and could not be recovered. The only option left was to check the files one by one.
After a thorough inspection of the whole project I recalled that: (1) my first attempts with the numerical simulations tried to use the more accurate McDonald-Johnson junction model [19], I later switched to the simplified Stewart-McCumber RSJ model because it was much faster and efficient in calculating the junction behavior [17, 18]; (2) file names attempted to reflect what the programs actually did, at least within the limits of DOS 8+3 naming scheme: in the same directory I could have a file ending with a “” that provided a textual output and another file ending with a “g” that gave a graphical output, and they differed only for a couple of `DEFINES`s that controlled the conditional compilation of the proper sections of the source code.
This multiplication of files might seem senseless today, when comfortable graphical interfaces, ultra-fast text editors with support of regular expressions and version control tools allow to change a large set of files in just a few seconds, but at that time it was probably the fastest, albeit very inefficient, way of working with source code; (3) the header of all source files contained detailed notes about the type of program, the compiler, the type of output and the dates of first and last revision of the source file, considerably

[2]The now defunct Digital Equipment Corporation (DEC) was one of the leading computer companies of the time and ULTRIX is the name of its Unix operating system.



```
c $DEFINE textout       ! DEFINEd for text output
CC $DEFINE graphout        ! DEFINEd for graphical output
c
c $DEFINE single        ! DEFINED for single rf drive
c $DEFINE biharmonic    ! DEFINED for biharmonic drive
CC $DEFINE pulsed       ! DEFINED for pulsed drive
c
c ======================================================================
c       Integration of RSJ Josephson junction model
c       with McCumber dimensionless parameters
c ----------------------------------------------------------------------
c       Name ......... MCPHASE.FOR (Phase Plot of McCumber RSJ model)
c       Program ...... Integration of Josephson equation with
c                      general rf drive (single, biharmonic or pulsed)
c       Output ....... Graphical
c       Version ...... 1.0
c       Language ..... Microsoft Fortran 5.1
c       Date ......... 06.10.1994
c       Last revision  08.10.1994
c       Author ....... S. Maggi
c ----------------------------------------------------------------------
c  Computes the I-V characteristic of a dc+rf current biased Josephson
c  junction, using the RSJ junction model and McCumber parameters.
c  The rf signal can be: single, biharmonic or pulsed.
c
c  The integration of the Josephson equation is performed by a 4th-order
c  Runge-Kutta method (Numerical Recipes).
c  The integration step is constant (=taustep).
c
c  The program computes the junction time response and, if the graphical
c  output is chosen,
c  (1) either plots the normalized voltage <eta>=<V>/(R*Ic)=d(phi)/d(tau)
c      vs. the normalized time tau, or phi vs. tau.
c  Otherwise, with textual output, the computed values of
c  <eta> vs. alpha.dc are printed on the screen.
```

Figure 1. Header of one of the Fortran source files. The left sidebar shows the directory structure of the project.

easing the analysis of the different versions of the Fortran programs (Fig. 1); (4) the initial versions of the Fortran programs were monolithic, a single source file contained the whole code, that consisted in about 1.000 lines of Fortran. Only at a later time, better computing practices taught me to divide the monolithic code into multiple source files, compiled and linked together with a `Makefile`.
Another invaluable tool to analyze the different versions of the source files was Meld, an open source application available for all major operating systems that can perform a two- and even a three-way comparison of files and directories. Using Meld I quickly realized that the two most interesting source files were `mcphase.for` and `mcp-work.for`, both located in the `mccumber/` directory (Fig. S1).
The first program, `mcphase.for`, simulated the behavior of the junction for a single value of α_{rf} read from the input configuration file `mc-iv.dat`, and saved the $I-V$ characteristic of the junction and its phase portrait (i.e., the relation between the phase and its time derivative, the latter being proportional to the junction voltage V) in the output file `mc-iv.out`.
Multiple calculations with several different value of α_{rf} were performed by using a DOS batch file that basically choose one by one the configuration files containing the desired values of α_{rf}, renamed them to `iv.dat`, run the compiled executable `mcphase.exe` and at the end renamed the file containing the results, `mc-iv.out`, using a consistent naming scheme. I don't recall why I choose this approach, but it was clearly very inefficient, as it required to prepare each day a long series of configuration files that differed only by the value of α_{rf}, and to update accordingly the DOS batch file that controlled the night calculations (Fig. S2).
The second program, `mcp-work.for` was an improved version that could cycle across a set of several values of α_{rf}, producing a different output file for each value of α_{rf}. To simplify the later automatic analysis, it left out the phase portrait.
Clearly this was the program ported to the DEC workstations. Unfortunately, I could not find the actual source file used on these machines, perhaps because I worked directly on the workstations and never thought to copy back these files to my PCs. But Fortran is a very stable language and making `mcp-work.for` work on a modern machine was very easy.

As for the Microsoft Visual Basic 1.0 code, I found only two versions of the programs and the differences between them were minimal. Since both programs gave exactly the same results, I decided to stick with the version that had a still working precompiled binary file.

4 Porting Microsoft Fortran to modern Unix

Porting `mcp-work.for` to the XXI century so that it could be compiled with the modern open source and multiplatform `gfortran` Fortran compiler was very easy, thanks to the stability of the language across different versions and platforms. Only a few minor tweaks to the source code were needed.
All work has been done on macOS, which is essentially BSD Unix with a more appealing graphical interface, but it can be easily repeated on any modern Unix-like operating system such as Linux, and probably even on Windows, with the support of either the Windows Subsystem for Linux (for Windows 10) or of Cygwin (for earlier versions of the operating system).

4.1 Preprocessor directives

For reasons that go beyond my understanding Microsoft Fortran 5.1 did not use standard preprocessor directives, such as those supported by `cpp` or `fpp`, [20] but used a a slightly different proprietary syntax (Fig. S3). To support `cpp`, all was needed was to comment out all the `$DEFINE` directives in the header section of `mcp-work.for` and to replace the Microsoft Fortran 5.1 `DEFINE` blocks with standard `cpp` blocks throughout the code (Fig. S3).
The right directives are chosen now at compile-time. For example, the following command [3]

```
$ gfortran -cpp -Dtextout -Dsingle -o mcp-work mcp-work.for
```

runs the `cpp` preprocessor before the gfortran compiler, selecting only the sections of code that produce a textual output (`-Dtextout`) and simulate the junction behavior with the sinusoidal drive (`-Dsingle`).

4.2 Filenames

Compilers based on Fortran 77, such as Microsoft Fortran 5.1, did not support dynamic memory allocation at runtime and required programmers to use fixed-length arrays and strings. Strings were used rather sparingly in Fortran code, so that was not a big deal. With an exception. My code defined the basename of all files as the 50-byte long character variable `filename`, attaching a proper extension to the input configuration file that contained the simulation parameters and to the output data files with the results of the calculations.
Under DOS that was not a problem, as DOS truncates file names to only 8 characters plus 3 characters for the extension, and excess characters were simply ignored. But under Unix file names have no practical limitations,[4] and having all these 50 character-long filenames, mostly composed by blank characters, was ugly and complicated file management, in particular when using the command line interface. The solution was simple, as Fortran now has the `TRIM` function, that removes all trailing blanks from a string. Whenever a file is opened for reading or for writing, `TRIM()` is applied on-the-fly to the `filename` variable,

[3]The $ symbol prepended to this and to all following terminal commands represents the prompt of the command interpreter and is not part of the command.
[4]Unix allows 255 characters for the filename and 4096 characters for the path.

```
    OPEN (UNIT = 10, FILE = TRIM(filename)//'.dat', STATUS = 'OLD')
```

thus removing all extra blanks from the name of the file.

4.3 Edit descriptors

Microsoft Fortran 5.1 used the backslash (\) edit descriptor to prevent the addition of a line break at the end of a `WRITE` instruction. Modern Fortran compilers do not support this non-standard edit descriptor and return an error. This problem is avoided by removing the backslash from all `WRITE` instructions that include it.

4.4 Date and time

Microsoft Fortran 5.1 had two separate intrinsic subroutines to return the current date and time. In particular, `CALL GETDAT(iyr, imon, iday)`, saved the date in the two-byte integer variables `iyr`, `imon` and `iday`, while `CALL GETTIM(ihr, imin, imin, i100th)` did the same for the current time, saving the return values in the integer variables `ihr`, `imin`, `imin` and `i100th`. The meaning of each returned variable should be self-explanatory.
Modern Fortran supports the single subroutine `CALL DATE_AND_TIME(DATE, TIME, ZONE, VALUES)`, where all arguments are optional and can be specified by their dummy names (i.e., how Fortran calls the keyword arguments of a function call). In particular, `DATE`, `TIME` and `ZONE` are character variables, while `VALUES` is a one-dimensional array of 8 integers, where `VALUES(1:3)` corresponds to the year, month and day of the month, `VALUES(4)` is the time difference (in minutes) with UTC, and `VALUES(5:8)` are the hour, minute, second and milliseconds, respectively.
To minimize changes to the original source code, the calls to the `GETDAT` and `GETTIM` subroutines, were translated to a single call to `DATE_AND_TIME`, assigning the elements of the returned array of `VALUES` to integer variables named as in the original code (Fig. S4).

4.5 Compilation with gfortran

As noted above, `mcp-work.for` calculates the $I-V$ characteristics of the simulated junction for several different values of α_{rf}, producing one output file for each $I-V$ curve. The section of the code that defined the names of the output files was quite convoluted,

```
      il=0
      DO alpha_rf=0.0, 50.0, 0.5
      ...
c ------ define output file name(s)
         il=il+1
         il2=INT(il/100)
         il1=INT(il/10)-il2*10
         il0=il-il1*10-il2*100
         filewrite='PU'//CHAR(il2+47)//CHAR(il1+47)//CHAR(il0+47)
      ...
      END DO         ! repeat alpha_rf DO cycle
```

and used the integer variable `il` to count the cycle number, while the three integers `il2`, `il1` and `il0` contained the hundreds, tens and units digits of `il`, respectively. The `CHAR` function converts these integers to the corresponding `ASCII` characters, where `ASCII` code 48 corresponds to the `0` symbol and `ASCII` code 57 corresponds to `9`. The name of the output file defined in the variable `filewrite` was built by concatenating a trailing constant string (`'PU'` in the example above) to the three `ASCII` characters, using the double forward slash (`//`) operator.

I have no idea why I decided to build the name the output files in such a complex way, while it would have been much simpler to use the value of α_{rf}. In any case, it did not work under gfortran and prevented proper compilation of the code.
After some inspection it was apparent that the number 47 added to integer variables in the three CHAR function calls was the source of the error, and that it should be replaced by 48. The line defining the `filewrite` variable thus becomes,

```
      ...
      DO alpha_rf=0.0, 50.0, 0.5
      ...
c ------ define output file name(s)
         ...
         filewrite='PU'//CHAR(il2+48)//CHAR(il1+48)//CHAR(il0+48)
      ...
      END DO        ! repeat alpha_rf DO cycle
```

With this change `mcp-work.for` could compile flawlessly under gfortran. What is still puzzling is how the original line could work in Microsoft Fortran 5.1.

5 Visual Basic code

No attempt was made to try to run the original Visual Basic 1.0 program on a modern computer. Visual Basic is a dead language and long since has been replaced by Visual Basic .NET, which shares only the name with its forefather.
From the beginning, the only viable option to run a Visual Basic 1.0 application today was to rebuild the original development environment based on DOS 6.22 and Windows 3.11 in an emulator. The other possible alternative, try to setup an ancient PC still capable to run DOS and Windows 3.11, albeit in principle interesting to ensure a replication of the original paper at the hardware level, would have posed more problems than it solved, adding little to the accuracy of the reproduction itself.
I preferred to use the Parallels Desktop emulator for macOS, but popular alternatives such as VMware Workstation for Windows or the VirtualBox open source multi-platform emulator should work equally well.
I created a new empty virtual machine with minimal hardware requirements and installed in sequence DOS 6.22, Windows 3.11 and Visual Basic 1.0 (Fig. 2). While I was at it, and although I had already reproduced the Fortran part of the project, I also decided to install Microsoft Fortran 5.1 for DOS, to try to recreate as much as possible the original work environment.
All software packages were downloaded from the WinWorld web site, an invaluable resource for recovering old software packages. Even after so many years, the legitimacy of installing proprietary software in an emulator, might be questionable. But at the time I had regular licenses for all the above mentioned software and I guess to be at least morally authorized to continue to use those packages. Unfortunately, this also means that it is not possible to share the image of the virtual machine used to run the Visual Basic program on the paper's GitHub repository, since it contains proprietary software.
The packages were originally distributed on several floppy disks, which had to be swapped whenever the installer required a new disk. The installation of these software packages in an emulator is close to how it was done back then. The only difference is that today the floppy disks are replaced by virtual file images and *swapping* disks is not done mechanically but requires to select a menu option in the emulator.
At the end of the installation process, the Windows 3.11 appeared as in Fig. 2. The default 640×480 pixel screen resolution of Windows 3.11 was woefully meager by today's standards but, as I used the virtual environment almost exclusively to run the Visual Basic program, I didn't bother to install the video drivers that could increase the screen resolution to a more comfortable 800×600 or 1024×768 pixel resolution.

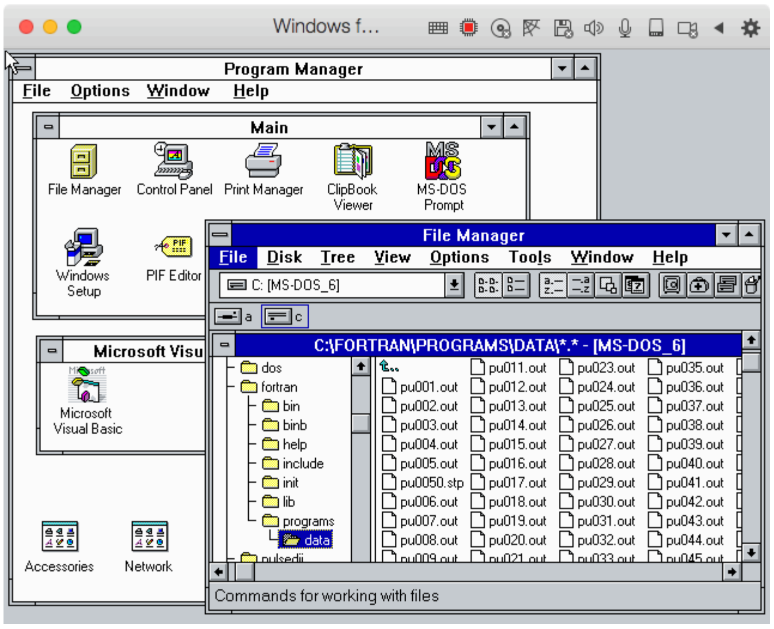

Windows f...
Program Manager
File Options Window Help
Main
File Manager
Control Panel
Print Manager
ClipBook Viewer
MS-DOS Prompt
Windows Setup
PIF Editor
Microsoft Visu
Microsoft Visual Basic
Accessories
Network
File Manager
File Disk Tree View Options Tools Window Help
C: [MS-DOS_6]
C:\FORTRAN\PROGRAMS\DATA\*.* - [MS-DOS_6]
dos
fortran
bin
binb
help
include
init
lib
programs
data
pulsedji
pu001.out pu002.out pu003.out pu004.out pu005.out pu0050.stp pu006.out pu007.out pu008.out pu009.out
pu011.out pu012.out pu013.out pu014.out pu015.out pu016.out pu017.out pu018.out pu019.out pu020.out pu021.out
pu023.out pu024.out pu025.out pu026.out pu027.out pu028.out pu029.out pu030.out pu031.out pu032.out pu033.out
pu035.out pu036.out pu037.out pu038.out pu039.out pu040.out pu041.out pu042.out pu043.out pu044.out pu045.out
Commands for working with files

Figure 2. The Windows 3.11 desktop as shown in the Parallels emulator. The File Manager window shows the files generated by the Fortran `mcp-work` program, before being processed by the Visual Basic program.

Using the emulator required to transfer the source and binary Visual Basic files to the emulated DOS/Windows system. It is surely possible to make the emulated Windows 3.11 communicate with the host operating system through the network. But I found much easier to use again a virtual floppy disk to transfer all needed files from macOS to Windows 3.11 (and viceversa).
A new empty virtual floppy disk `data.img` can be easily created with the command-line utility dd available on macOS or Linux

```
dd if=/dev/zero of=data.img bs=1440k count=1
```

After creation, the virtual floppy disk must be mounted in the emulator and formatted under DOS or Windows 3.11 in the original MS-DOS FAT file system.
This step completed the preparation of the development environment, now it was time to test how all this behaved.

6 Running the programs

As already noted, `mcp-work.for` calculates the $I - V$ characteristics for a range of values of α_{rf}, saving each curve in a separate output file. The lower and upper limits and the step size of α_{rf} are hardcoded in the Fortran source code ad every change requires a recompilation of `mcp-work.for` (a minor hassle, as the compilation takes just a couple of seconds on a modern machine).
Also the names of the output data files are hard-coded in `mcp-work.for` in the `filewrite` variable and are conventionally composed by a two-letter prefix ("SI" for the single drive, "BI" for the biharmonic drive and "PU" for the pulsed drive) followed by a three-digit integer that represents the cycle number (Section 4.5).
To avoid cluttering the `mccumber/` directory that contains the Fortran source files with the output data files produced by the simulations, I created a new directory in the main project folder, `2020runs/`, where I copied the `mc-iv.dat` configuration file needed to start the simulation (Fig. 1).

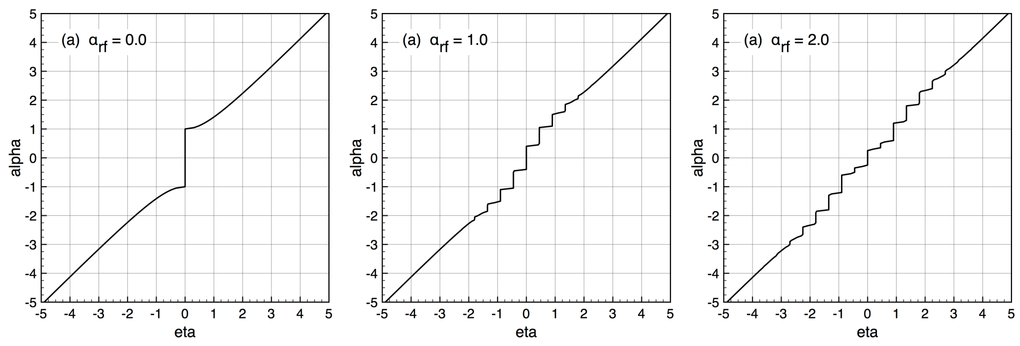


Figure 3. $I - V$ characteristics of a junction with $\beta_c = 0.01$, driven by a sinusoidal microwave signal of frequency $\Omega = 0.45$ and (a) $\alpha_{\mathrm{rf}} = 0.0$, (b) $\alpha_{\mathrm{rf}} = 1.0$ and (c) $\alpha_{\mathrm{rf}} = 2.0$. The rf-induced current steps are clearly visible when $\alpha_{\mathrm{rf}} > 0$. Here `eta` and `alpha` are the normalized voltage η and normalized current α, respectively.

Running `mcp-work.for` requires three steps: compile `mcp-work.for` with the proper directives, switch to the `2020runs/` directory and run the `mcp-work` executable from there. The whole process is summarized below for the sinusoidal drive

```
$ gfortran -cpp -Dtextout -Dsingle -o mcp-work mcp-work.for
$ cd ../2020runs/
$ ../mccumber/mcp-work
```

The only modification needed to perform calculations using the pulsed drive is to change the `-Dsingle` directive to `-Dpulsed`

```
$ gfortran -cpp -Dtextout -Dpulsed -o mcp-work mcp-work.for
$ cd ../2020runs/
$ ../mccumber/mcp-work
```

On a recent (but not state-of-the art) machine the whole simulation with 100 α_{rf} steps takes around 5 minutes for the single drive and 7 minutes for the pulsed drive, and most of the time is spent printing on the terminal the calculated $I-V$ characteristics for each value of α_{rf}. Such feedback was useful at the time of the original calculation, as every new calculated point of the $I - V$ characteristics appeared on the screen after several tens of seconds, now the results scroll on the screen at a speed that makes them almost illegible. However, to keep as faithful as possible to the original project, I decided to continue to print the data points on the computer screen.
The $I-V$ characteristics calculated with the sinusoidal drive are shown in Fig. 3 for different values of α_{rf}. Without microwave radiation ($\alpha_{\mathrm{rf}} = 0$), the simulation produces the well-known $I - V$ characteristic of an overdamped Josephson junction (Fig. 3a), while for non-zero values of α_{rf} the staircase-like structure of the rf-induced current steps appears on the $I - V$ curves (Fig. 3b and Fig. 3c).
For a pulsed drive, the $I - V$ characteristic without microwave radiation is identical to that calculated with the sinusoidal signal (Fig. 4a), while for $\alpha_{\mathrm{rf}} > 0.0$ the current steps induced by the pulsed drive are fewer than with the sinusoidal drive and can be nearly as wide as the critical current (compare Figs. 4a and 4c).
At the end of each run, the output files should be transferred to the Windows 3.11 virtual machine to be processed by `stepampl`, the Visual Basic application described in Section 5. However Windows 3.11 does not *understand* Unix line terminators and cannot read the output files without a preliminary conversion. The conversion can be easily done in the macOS Terminal by issuing the following command,

```
$ for f in $(ls *.out); do sed -i .bak s/$/$'\r'/ $f ; done
```

that changes the line terminators of the `.out` output files from the Unix format containing only a line-feed (LF) to the carriage return followed by a line-feed (CR-LF) format

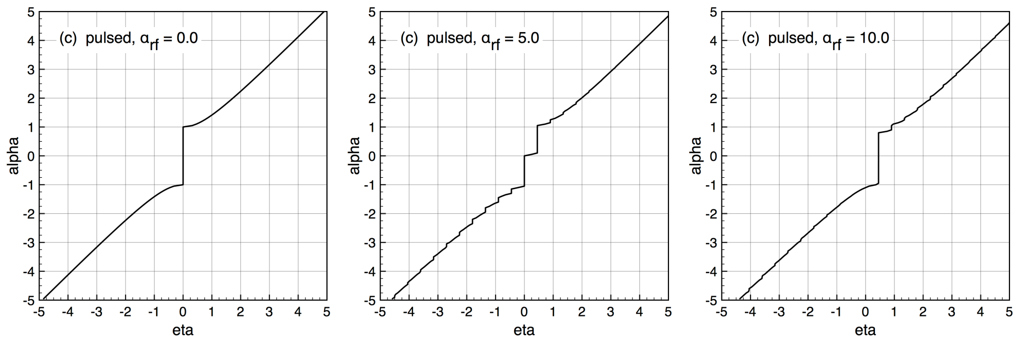


Figure 4. $I - V$ characteristics of a junction with $\beta_c = 0.01$, driven by a pulsed microwave signal of frequency $\Omega = 0.45$ for: (a) $\alpha_{\mathrm{rf}} = 0.0$, (b) $\alpha_{\mathrm{rf}} = 5.0$ and (c) $\alpha_{\mathrm{rf}} = 10.0$. For $\alpha_{\mathrm{rf}} > 0$ the rf-induced current steps are much larger than with the standard sinusoidal drive. Here `eta` and `alpha` are the normalized voltage η and normalized current α, respectively.

used by all versions of Microsoft Windows.[5]. The `-i` switch allows in-place conversion of each file. The original files are kept adding a `.bak` extension.
The output files in the proper Windows compatible format can now be transferred onto the virtual floppy disk image. When the transfer is done, the floppy disk image is unmounted from the host operating system and mounted in the virtual machine, making it visible to Windows 3.11. The output files are copied to an empty directory of Windows 3.11 and the Visual Basic application is started, either by running the precompiled `stepampl.exe` executable or by opening the Visual Basic project and running the program from there (Fig. S5), saving the results in another text file with extension `.STP` (for steps) that could be transferred back to the host operating system via the virtual floppy disk image.
I also briefly tried to run the original Fortran code using the Microsoft Fortran 5.1 installed in the emulator. Compilation was fine but the resulting DOS program was extremely slow, taking about $30 - 35$ seconds for each α_{rf} cycle and about 60 minutes in total for the sinusoidal drive, more than a tenfold increase with respect to the native macOS version compiled with gfortran. Even considering the overhead of the emulator, the difference is too large not to be attributed to the low quality of the binary code generated by the Microsoft Fortran 5.1 compiler.

7 Results

At the time of writing the original paper the whole process had to be repeated each night for a different set of input parameters and for a different kind of microwave signal (single, biharmonic or pulsed).
Each night I used three of the available DEC workstations to simulate the junction behavior with the pulsed drive, using three different values of the (normalized) width of the pulse signal, ρ, while keeping constant all the other parameters, such β and Ω. The only other difference in these simulation was the range of variation of α_{rf}, which depended on the value of ρ (shorter pulses require a much larger intensity of the rf-signal to have the same effect on the junction). The fourth workstation simulated the junction behavior with the standard sinusoidal drive, using exactly the same set of junction parameters.
The original paper contained all the information needed to reproduce the results shown in the figures, without having to repeat the whole analysis from scratch. The parameters used in all runs were: hysteresis parameter $\beta = 0.01$, frequency of the sinusoidal or

[5]Slightly different versions of the command can be found on the internet; the format used above is POSIX-compliant and should run on any Unix flavour

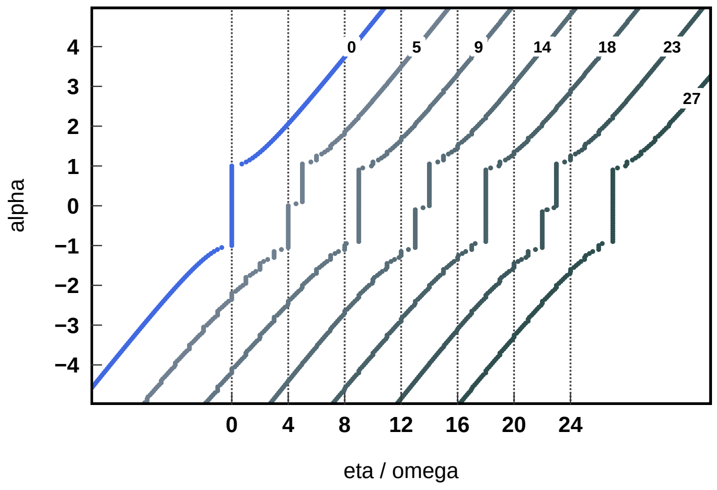


Figure 5. $I - V$ characteristics of a junction with $\beta = 0.01$ for several values of α_{rf}. The junction is irradiated by a train of pulses of repetition frequency $\Omega = 0.45$ and width $\rho = 0.05$. Each curve is offset horizontally by 4Ω and is labelled after the value of α_{rf}. The vertical dotted lines mark the position of the zero-voltage axis (i.e., the critical current I_C) for the $I - V$ curve located immediately to the right.

pulsed rf signal $\Omega = 0.45$, current bias between $\alpha_{\mathrm{dc}} = -5.0$ and $\alpha_{\mathrm{dc}} = 5.0$, integration time $\tau = 500$, time step $\Delta\tau = 0.01$.
The simulation with the sinusoidal drive was performed by varying the amplitude of the microwave signal α_{rf} between 0.0 and 5.0, with a step $\Delta\alpha_{\mathrm{rf}} = 0.05$. The three simulations with the pulsed drive were done using: (1) pulse width $\rho = 0.250$, $\alpha_{\mathrm{rf}} = 0.0 - 10.0$, $\Delta\alpha_{\mathrm{rf}} = 0.1$; (2) pulse width $\rho = 0.125$, $\alpha_{\mathrm{rf}} = 0.0 - 20.0$, $\Delta\alpha_{\mathrm{rf}} = 0.2$; (3) pulse width $\rho = 0.050$, $\alpha_{\mathrm{rf}} = 0.0 - 50.0$, $\Delta\alpha_{\mathrm{rf}} = 0.5$.
The resulting output and summary files were saved in separate folders in the `2020runs/` directory, named `SINGLE/`, `PULS0250/`, `PULS125/`, `PULS0050/` after their DOS counterparts.
To reproduce the first and second figure of Ref. [14] I made the only concession to modernity. Instead of trying to recreate them with the plotting program used originally, probably Origin 2.0, I decided to write a couple of small R scripts that could automate the task. The results are shown in Fig. 5 and Fig. 6 and, as expected, are identical to those reported in the first two figures of Ref. [14]. The large vertical steps on the rightmost curves of Fig. 5 era the first rf-induced current steps, that can be nearly as large as the critical current I_C without rf bias visible in the first $I - V$ characteristic on the left.
Figure 3 of the original paper could also be easily reproduced by plotting the maxima of the curves of Fig. 6, i.e., Δi_n vs. α_{rf}, for the four different cases considered here.
Similar considerations can be made for the reproduction of Figure 4 of the original paper, which considers a slightly hysteretic junction with $\beta = 1.0$. For simplicity, I have chosen to show instead the Δi_n vs. α_{rf} curves for the two most significant cases of microwave signal, i.e., the sinusoidal drive and the pulsed drive with $\rho = 0.050$ (Fig. 7), from which the plots of Figure 4 could be easily replicated.

8 Code availability

All code used for this replication is available in the project's GitHub repository. However, the term "all" should be taken with a grain of salt. While the Fortran is truly available to everyone and can be used as-is by compiling it with gfortran or with any other compatible modern Fortran compiler, the Visual Basic 1.0 code poses a completely different set

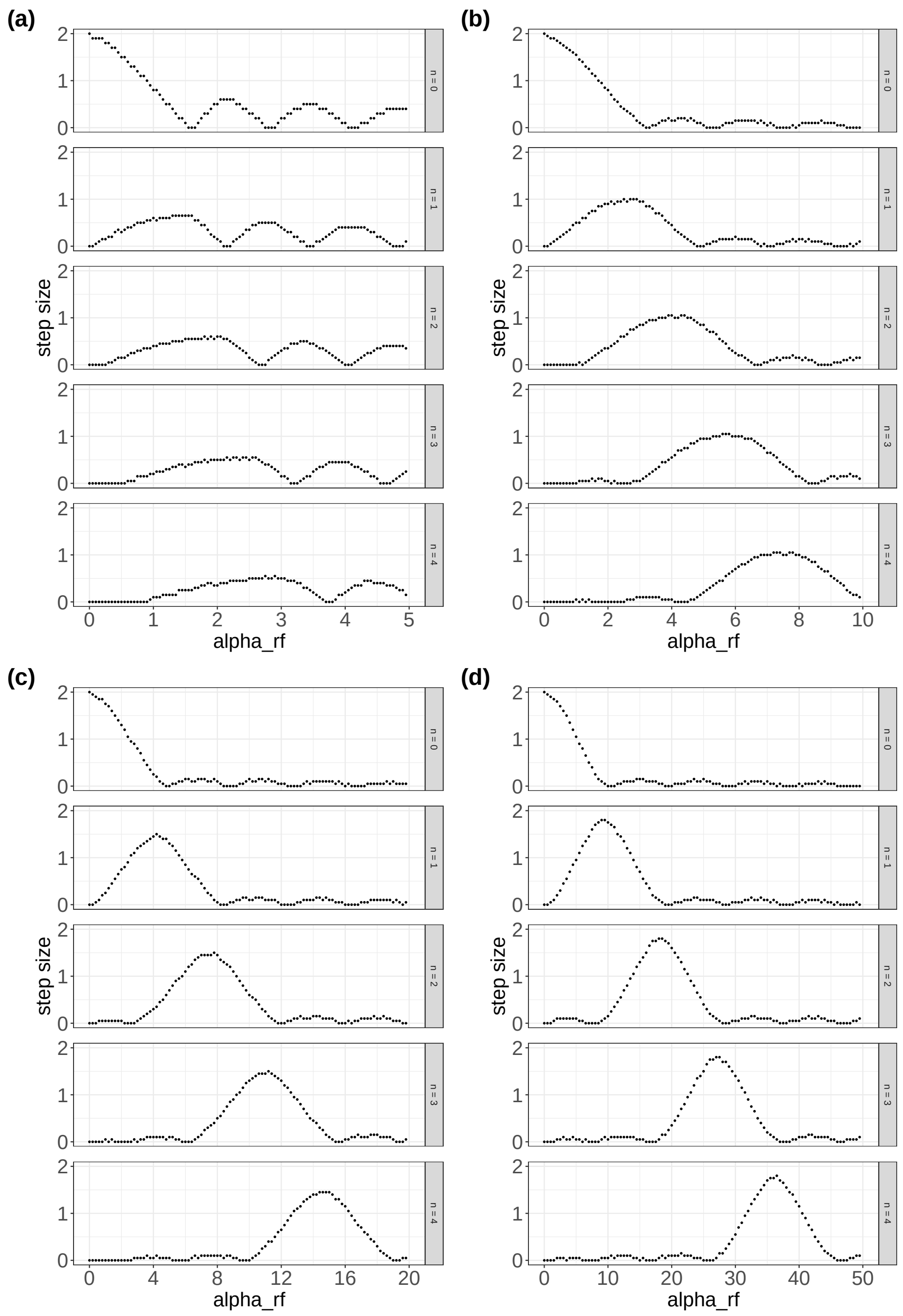


Figure 6. Dependence of the step size Δi_n on the amplitude of α_{rf}, for $n = 0...4$. The $n = 0$ step is the normalized total critical current Δi_0 of the junction. The junction parameters are $\beta = 0.01$ and $\Omega = 0.45$; (a) sinusoidal drive, (b) pulsed drive with $\rho = 0.250$, (c) pulsed drive with $\rho = 0.125$, (d) pulsed drive with $\rho = 0.050$.

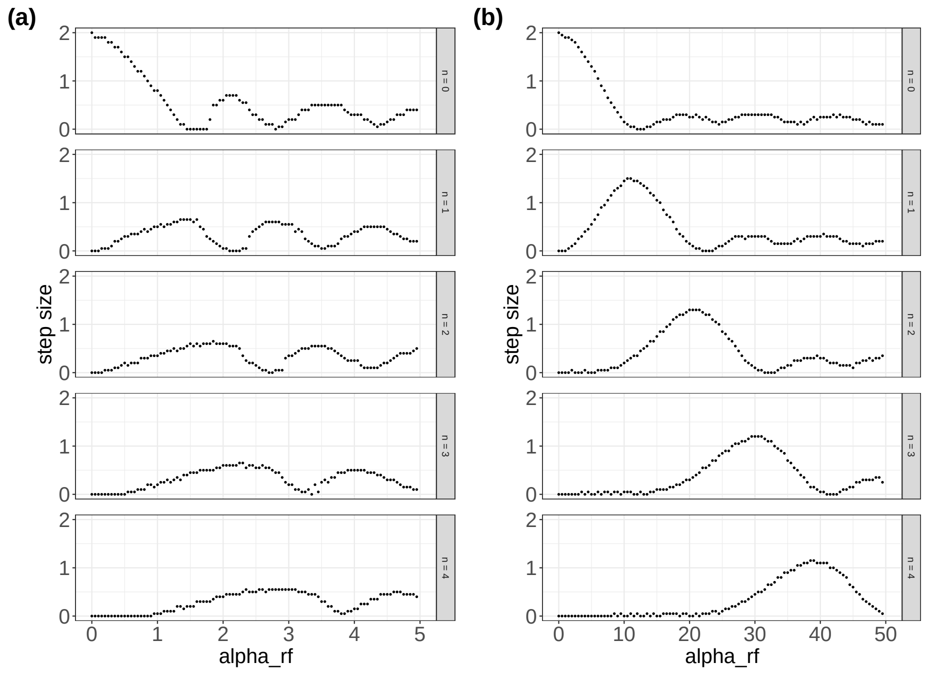


Figure 7. Dependence of the step size Δi_n on the amplitude of α_{rf}, for $n = 0...4$. The $n = 0$ step is the normalized total critical current Δi_0 of the junction. The junction parameters are $\beta = 1.0$ and $\Omega = 0.45$; (a) sinusoidal drive, (b) pulsed drive with $\rho = 0.050$.

of problems.
First of all, it can be run only by rebuilding an exact replica of the original development environment, as noted in Section 5. A task that without proper documentation can require a lot of trial-and error, as the author of the present paper discovered during the course of this work, when he found that the original code could not be imported in any later versions of Visual Basic for Windows and even in Visual Basic 1.0 for MS-DOS, released by Microsoft in parallel with the initial Windows version.
Second, the source code of a Visual basic program is a mix of Basic source files (having the usual `.bas` extension) and of Forms objects that compose the graphical interface of the program (with extension `.frm`), stored in some binary format and tied together in a container, known as a Visual Basic project (extension `.mak`), as shown in Fig. S6). Making things worse, a Visual Basic project cannot be exported, the idea of code sharing or reuse across different applications was almost unknown at the time, at least in the commercial world.
The obvious consequence is that the reproduction described here would fail to comply, at least in part, with one of the basic requirements of the Ten Years Challenge, i.e. the availability of all the source code used for the reproduction.
But even if Visual Basic code seems locked for eternity in its proprietary binary format, there is a simple, albeit partial, solution to this problem, printing to a file. Printing was a true necessity back then to inspect or debug code. Computer screens were small and editors were primitive, the best way to have an overall view of a program during development was to print it on paper. Visual Basic 1.0 is no exception and can easily print both the Basic source code and the layout of each form composing the graphical interface. Once realized that, it is easy to set up a PostScript Printer in Windows 3.11 and to print the Basic sources and the forms to two separate PostScript files on the Windows 3.11 virtual disk. Transfer of these files to macOS is done again using the virtual floppy disk image and, once in macOS, it is a questions of seconds to convert the PostScript files to the more popular PDF format using the Preview application available in every version of

macOS. The finishing touch is to use an R script (already developed for another project) to extract the text from the PDF file containing the Basic code, saving it to a true text source file, so that it can be easily inspected by everyone interested in this project.

9 Discussion

The main problem with this reproduction was the accidental loss of all my handwritten notes about the project, that compelled me to recover all information needed from the source and data files. Keeping good and updated documentation about any research project is paramount but equally important is to store all documentation in a safe location, where it can be easily retrieved.
Today most documents are in electronic form, making them even more prone to data loss whenever a disaster strikes. Implementing a good and reliable backup strategy on different forms of storage media should be a mandatory requirement of any research project, and this strategy should always combine local backups with backups on secondary storage locations, physically well separated from the original data source.
Starting the project today from scratch I would make very different choices about the programming languages to use for the simulation and data analysis steps.
Fortran, despite its venerable age, is still an excellent language for scientific programming, but today I would surely prefer Python, because of its flexibility, ease of use, and availability of excellent numerical libraries, such as NumPy and SciPy.
The main problem with Python is related to the tumultuous development of the language itself and of the thousands of available modules, which can cause incompatibilities even with code developed a few years ago. This problem can be, at least temporarily, be solved by using virtual environments, but a better standardized solution is strongly needed.
Another problem is related to its nature of interpreted language. However, the presence of well-documented Fortran and C bindings, can enhance performance of time-critical sections of code.
Starting today I would also avoid using a new language, as was Visual Basic 1.0 at the time for a scientific project. It is true that in 1993-94 it would have been very hard to foresee the rapid demise of Microsoft's Visual Basic, nevertheless using a programming language only when its main core is stable, runs on a wide array of operating systems and is accepted by a wide community of developers is surely a safer bet.
Not every software project can afford to be as stable as $\TeX$, the scientific typesetting system invented by the prominent mathematician and computer scientist Donald E. Knuth, that has reached a state where "it is unwise to make further improvements to the system [..] which should give the same results 100 years from now that they produce today" [21]. But on the other hand, a development environment that changes too much and too often or that is subjected to the whims of a single software company creates more problems than it solves.

10 Conclusions

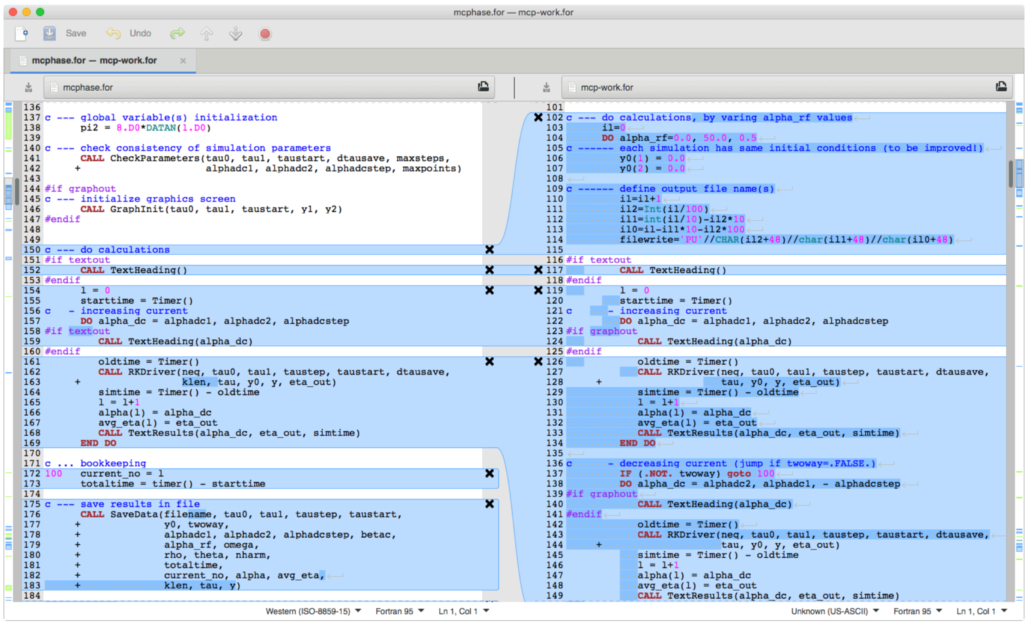
Going back to my old paper has been an exceptionally interesting and instructive experience and I thank the organizers of this challenge for the opportunity offered.
However this is not only a nostalgic attitude. The reproducibility crisis is a serious issue today [22], that undermines scientific credibility and impacts the public's trust in science, paving the way to all sorts of fake and unscientific beliefs. Being able to go back and reproduce what has been done in the past could ease the retraction of published papers containing fabricated, falsified, or modified data or results and could contribute to simplify the identification of future frauds.

This of course requires to share and make freely available the original data and the tools used to analyze them. A few years ago this requirement was impossible to fulfill in practice. The digital world in which we live makes it almost inevitable.

References

1. A. Barone and G. Paternò. **Physics and Applications of the Josephson Effect**. Wiley, July 1982.
2. M. Gurvitch, M. A. Washington, and H. A. Huggins. "High quality refractory Josephson tunnel junctions utilizing thin aluminum layers." In: **Applied Physics Letters** 42.5 (Mar. 1983), pp. 472–474.
3. S. P. Benz. "Superconductor normal superconductor junctions for programmable voltage standards." In: **Applied Physics Letters** 67.18 (Oct. 1995), pp. 2714–2716.
4. S. A. Cybart, E. Y. Cho, T. J. Wong, B. H. Wehlin, M. K. Ma, C. Huynh, and R. C. Dynes. "Nano Josephson superconducting tunnel junctions in YBa2Cu3O7$-\delta$ directly patterned with a focused helium ion beam." In: **Nature Nanotechnology** 10.7 (July 2015), pp. 598–602.
5. N. De Leo, M. Fretto, V. Lacquaniti, C. Cassiago, L. D'Ortenzi, L. Boarino, and S. Maggi. "Thickness Modulated Niobium Nanoconstrictions by Focused Ion Beam and Anodization." In: **IEEE Transactions on Applied Superconductivity** 26.3 (Apr. 2016), pp. 1–5.
6. K. Likharev and V. Semenov. "RSFQ logic/memory family: a new Josephson-junction technology for sub-terahertz-clock-frequency digital systems." In: **IEEE Transactions on Appiled Superconductivity** 1.1 (Mar. 1991), pp. 3–28.
7. S. Maggi, N. De Leo, V. Lacquaniti, A. Agostino, R. Gonnelli, and P. Verhoeve. "Nb/Al STJ detectors with sub-nA subgap current." In: **Physica C: Superconductivity and its Applications** 435.1-2 (Mar. 2006), pp. 103–106.
8. A. G. P. Troeman, H. Derking, B. Borger, J. Pleikies, D. Veldhuis, and H. Hilgenkamp. "NanoSQUIDs Based on Niobium Constrictions." In: **Nano Letters** 7.7 (2007), pp. 2152–2156.
9. C. Granata, A. Vettoliere, R. Russo, M. Fretto, N. D. Leo, E. Enrico, and V. Lacquaniti. "Ultra High Sensitive Niobium NanoSQUID by Focused Ion Beam Sculpting." In: **Journal of Superconductivity and Novel Magnetism** 28.2 (Feb. 2015), pp. 585–589.
10. R. Kautz, C. Hamilton, and F. Lloyd. "Series-array Josephson voltage standards." In: **IEEE Transactions on Magnetics** 23.2 (Mar. 1987), pp. 883–890.
11. R. Monaco. "Enhanced ac Josephson effect." In: **Journal of Applied Physics** 68.2 (July 1990), pp. 679–687.
12. D. Andreone, V. Lacquaniti, and S. Maggi. "Experiments on Josephson Junctions Driven by a Bi-Harmonic RF Source." In: **Nonlinear Superconductive Electronics and Josephson Devices**. Boston, MA: Springer US, 1991, pp. 37–43.
13. D. Andreone, V. Lacquaniti, and S. Maggi. "Numerical and Experimental Results on Josephson Junctions Irradiated by a Biharmonic Drive." In: **Superconducting Devices and Their Applications**. 1992, pp. 399–402.
14. S. Maggi. "Step width enhancement in a pulse driven Josephson junction." In: **Journal of Applied Physics** 79.10 (May 1996), pp. 7860–7863.
15. S. Maggi. "Enhanced phase locking in a Josephson junction driven by current pulses." In: **Journal of Low Temperature Physics** 106.3-4 (Feb. 1997), pp. 399–404.
16. R. W. Henry and D. E. Prober. "Electronic analogs of double junction and single junction SQUIDs." In: **Review of Scientific Instruments** 52.6 (June 1981), pp. 902–914.
17. D. E. McCumber. "Effect of ac Impedance on dc Voltage Current Characteristics of Superconductor Weak Link Junctions." In: **Journal of Applied Physics** 39.7 (June 1968), pp. 3113–3118.
18. W. C. Stewart. "Current voltage characteristics of superconducting tunnel junctions." In: **Journal of Applied Physics** 45.1 (Jan. 1974), pp. 452–456.
19. D. G. McDonald, E. G. Johnson, and R. E. Harris. "Modeling Josephson junctions." In: **Physical Review B** 13.3 (Feb. 1976), pp. 1028–1031.
20. A. Boyanski. **FPP - A Fortran Preprocessor**. Tech. rep. Department of Energy, 1992, pp. 1–7.
21. D. E. Knuth. "The Future of Tex and METAFONT." In: **TUGboat** 11.4 (Dec. 1990), p. 489.
22. T. Miyakawa. "No raw data, no science: another possible source of the reproducibility crisis." In: **Molecular Brain** 13.1 (Feb. 2020), pp. 1–6.

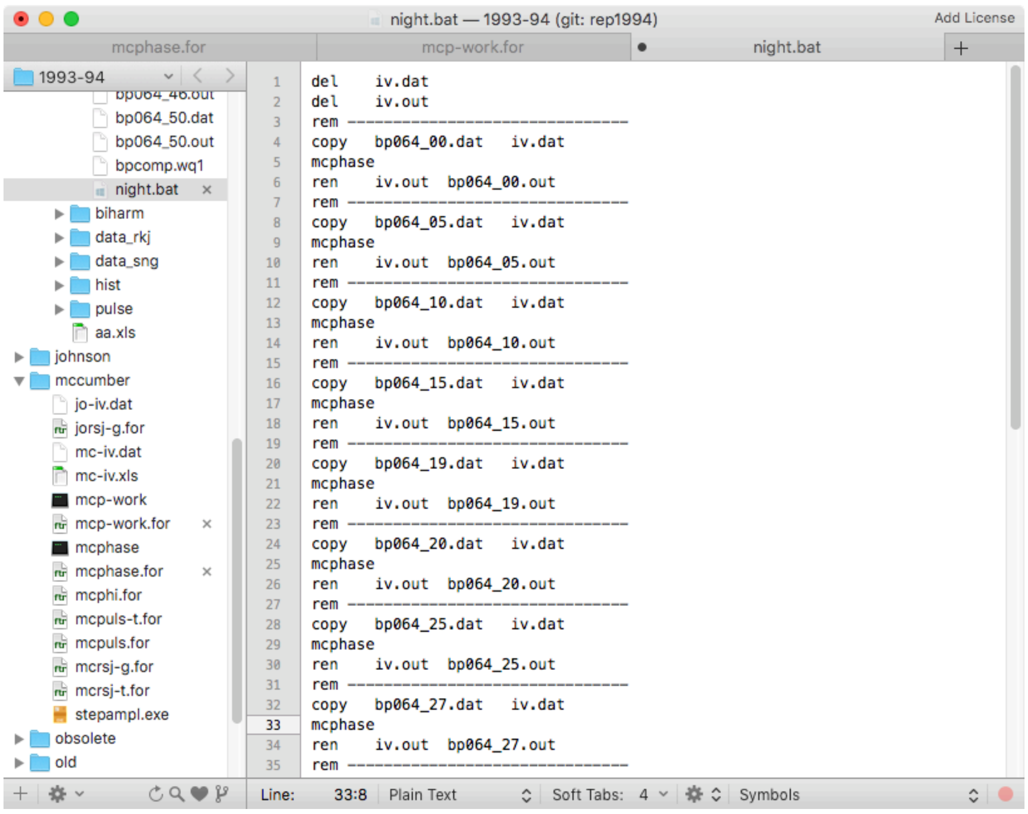
S1 Supplementary Material



```fortran

c --- global variable(s) initialization
      pi2 = 8.D0*DATAN(1.D0)

c --- check consistency of simulation parameters
      CALL CheckParameters(tau0, tau1, taustart, dtausave, maxsteps,
     +                     alphadc1, alphadc2, alphadcstep, maxpoints)

#if graphout
c --- initialize graphics screen
      CALL GraphInit(tau0, tau1, taustart, y1, y2)
#endif


c --- do calculations
#if textout
      CALL TextHeading()
#endif
      l = 0
      starttime = Timer()
c   - increasing current
      DO alpha_dc = alphadc1, alphadc2, alphadcstep
#if textout
         CALL TextHeading(alpha_dc)
#endif
         oldtime = Timer()
         CALL RKDriver(neq, tau0, tau1, taustep, taustart, dtausave,
     +                 klen, tau, y0, y, eta_out)
         simtime = Timer() - oldtime
         l = l+1
         alpha(l) = alpha_dc
         avg_eta(l) = eta_out
         CALL TextResults(alpha_dc, eta_out, simtime)
      END DO

c ... bookkeeping
100   current_no = l
      totaltime = timer() - starttime

c --- save results in file
      CALL SaveData(filename, tau0, tau1, taustep, taustart,
     +              y0, twoway,
     +              alphadc1, alphadc2, alphadcstep, betac,
     +              alpha_rf, omega,
     +              rho, theta, nharm,
     +              totaltime,
     +              current_no, alpha, avg_eta,
     +              klen, tau, y)

```

```fortran

c --- do calculations, by varing alpha_rf values
      il=0
      DO alpha_rf=0.0, 50.0, 0.5
c ------ each simulation has same initial conditions (to be improved!)
         y0(1) = 0.0
         y0(2) = 0.0

c ------ define output file name(s)
         il=il+1
         il2=Int(il/100)
         il1=int(il/10)-il2*10
         il0=il-il1*10-il2*100
         filewrite='PU'//CHAR(il2+48)//char(il1+48)//char(il0+48)

#if textout
         CALL TextHeading()
#endif
         l = 0
         starttime = Timer()
c        - increasing current
         DO alpha_dc = alphadc1, alphadc2, alphadcstep
#if graphout
            CALL TextHeading(alpha_dc)
#endif
            oldtime = Timer()
            CALL RKDriver(neq, tau0, tau1, taustep, taustart, dtausave,
     +                       tau, y0, y, eta_out)
            simtime = Timer() - oldtime
            l = l+1
            alpha(l) = alpha_dc
            avg_eta(l) = eta_out
            CALL TextResults(alpha_dc, eta_out, simtime)
         END DO

c        - decreasing current (jump if twoway=.FALSE.)
         IF (.NOT. twoway) goto 100
         DO alpha_dc = alphadc2, alphadc1, - alphadcstep
#if graphout
            CALL TextHeading(alpha_dc)
#endif
            oldtime = Timer()
            CALL RKDriver(neq, tau0, tau1, taustep, taustart, dtausave,
     +                       tau, y0, y, eta_out)
            simtime = Timer() - oldtime
            l = l+1
            alpha(l) = alpha_dc
            avg_eta(l) = eta_out
            CALL TextResults(alpha_dc, eta_out, simtime)
```

Figure S1. Comparison with Meld of the main calculation loop of (left) `mcphase.for` and (right) `mcp-work.for`.



```
del    iv.dat
del    iv.out
rem -------------------------------
copy   bp064_00.dat   iv.dat
mcphase
ren    iv.out  bp064_00.out
rem -------------------------------
copy   bp064_05.dat   iv.dat
mcphase
ren    iv.out  bp064_05.out
rem -------------------------------
copy   bp064_10.dat   iv.dat
mcphase
ren    iv.out  bp064_10.out
rem -------------------------------
copy   bp064_15.dat   iv.dat
mcphase
ren    iv.out  bp064_15.out
rem -------------------------------
copy   bp064_19.dat   iv.dat
mcphase
ren    iv.out  bp064_19.out
rem -------------------------------
copy   bp064_20.dat   iv.dat
mcphase
ren    iv.out  bp064_20.out
rem -------------------------------
copy   bp064_25.dat   iv.dat
mcphase
ren    iv.out  bp064_25.out
rem -------------------------------
copy   bp064_27.dat   iv.dat
mcphase
ren    iv.out  bp064_27.out
rem -------------------------------
```

Figure S2. DOS batch file used by `mcphase.for` to run multiple simulations with different values of the amplitude of the microwave signal α_{rf}.

(a)

```
$DEFINE textout           ! DEFINEd for text output
c $DEFINE graphout        ! DEFINEd for graphical output

c $DEFINE single        ! DEFINED for single rf drive
c $DEFINE biharmonic    ! DEFINED for biharmonic drive
$DEFINE pulsed            ! DEFINED for pulsed drive
...
...
$if defined (graphout)
  ...
$endif
...
...
$if defined (textout)
  ...
$endif
...
...
$if defined (pulsed)
  ...
$endif
...
```

(b)

```
CC $DEFINE textout        ! DEFINEd for text output
c $DEFINE graphout        ! DEFINEd for graphical output

c $DEFINE single        ! DEFINED for single rf drive
c $DEFINE biharmonic    ! DEFINED for biharmonic drive
CC $DEFINE pulsed     ! DEFINED for pulsed drive
...
...
#if graphout
  ...
#endif
...
...
#if textout
  ...
#endif
...
...
#if pulsed
  ...
#endif
...
```

Figure S3. Preprocessor directives in (a) Microsoft Fortran 5.1, (b) modern cpp preprocessor.

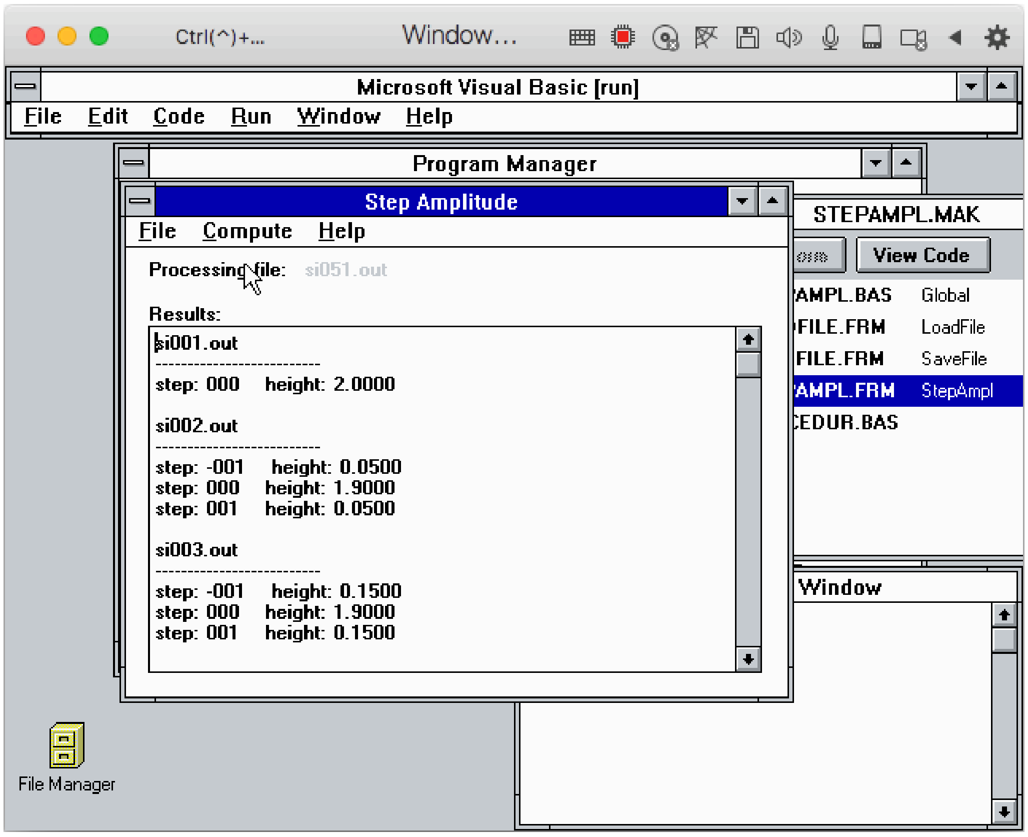
(a)

```
...
INTEGER*2      iyr, imon, iday
INTEGER*2      ihr, imin, isec, dummy
...
...
CALL GETDAT(iyr, imon, iday)
CALL GETTIM(ihr, imin, isec, dummy)
...
```

(b)

```
...
character*8    date
character*10   time
character*5    zone
integer        values(8)
...
...
call date_and_time(date, time, zone, values)
...
...
iyr  = values(1)
imon = values(2)
iday = values(3)
ihr  = values(5)
imin = values(6)
isec = values(7)
...
```

Figure S4. Getting the date and time in (a) Microsoft Fortran 5.1, (b) modern gfortan.

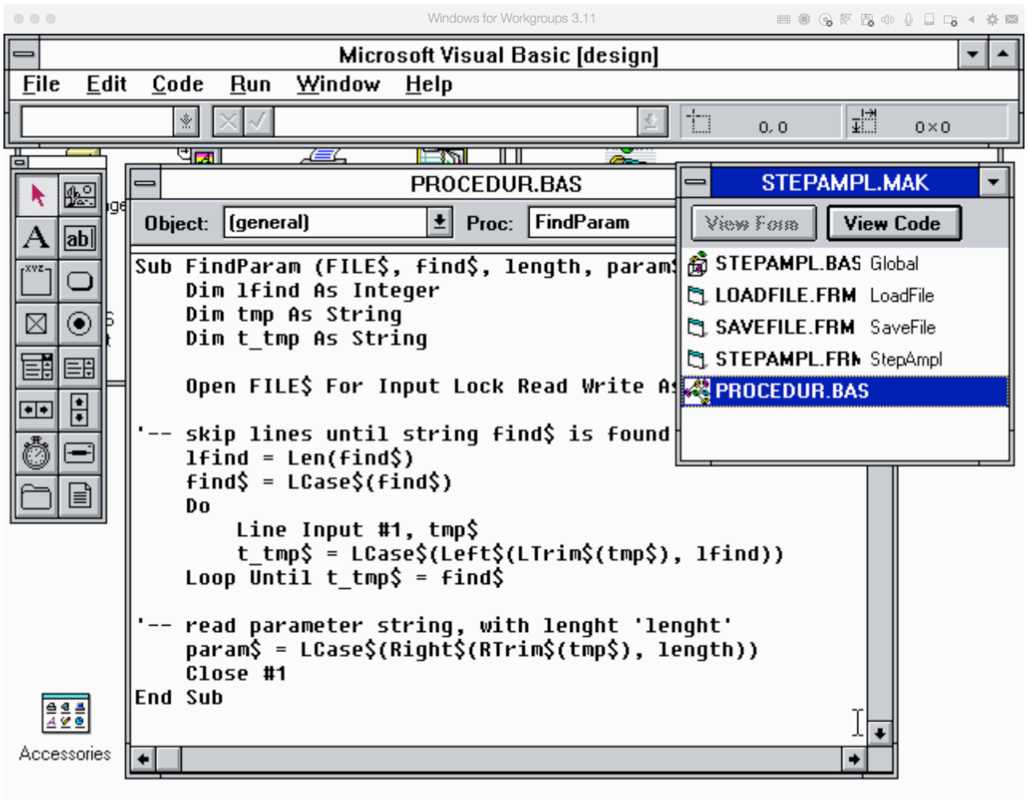


Processing file: si051.out

Results:

```
si001.out
--------------------------
step: 000    height: 2.0000

si002.out
--------------------------
step: -001    height: 0.0500
step: 000    height: 1.9000
step: 001    height: 0.0500

si003.out
--------------------------
step: -001    height: 0.1500
step: 000    height: 1.9000
step: 001    height: 0.1500
```

Figure S5. Windows 3.11 desktop with the running Visual Basic program `stepampl`.



```
Sub FindParam (FILE$, find$, length, param$
    Dim lfind As Integer
    Dim tmp As String
    Dim t_tmp As String

    Open FILE$ For Input Lock Read Write As

'-- skip lines until string find$ is found
    lfind = Len(find$)
    find$ = LCase$(find$)
    Do
        Line Input #1, tmp$
        t_tmp$ = LCase$(Left$(LTrim$(tmp$), lfind))
    Loop Until t_tmp$ = find$

'-- read parameter string, with lenght 'lenght'
    param$ = LCase$(Right$(RTrim$(tmp$), length))
    Close #1
End Sub
```

STEPAMPL.BAS	Global
LOADFILE.FRM	LoadFile
SAVEFILE.FRM	SaveFile
STEPAMPL.FRM	StepAmpl
PROCEDUR.BAS	

Figure S6. Visual Basic 1.0 development environment. The window in the foregound lists all the Basic files (with extension .bas) and Forms objects (extension .frm) that compose the Visual Basic project (extension .mak). An example of Visual Basic code is visible in the window in the background.